

AMD 64-Bit Technology

The AMD x86-64 Architecture Programmers Manual

Volume 3

Instruction-Set Reference



Publication # 24594 Rev: 2.90 Issue Date: November 2001
--

© 2001 Advanced Micro Devices, Inc. All rights reserved.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Trademarks

AMD, the AMD logo, AMD Athlon, AMD Duron, AMD-K5, 3DNow!, and combinations thereof, and Am486 and Am5x86 are trademarks, and AMD-K6 is a registered trademark, of Advanced Micro Devices, Inc.

MMX is a trademark and Pentium is a registered trademark of Intel Corporation.

Windows NT is a trademark of Microsoft Corp.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Contents

Contents	.iii
Figures	xv
Tables	xvii
Preface	xxi
Audience	xxi
Organization	xxi
Definitions	xxi
Related Documents	xxxiii
1 Instruction Formats	1
1.1 Instruction Byte Order	1
1.2 Instruction Prefixes	3
Summary of Legacy Prefixes	3
Operand-Size Override Prefix	5
Address-Size Override Prefix	6
Segment-Override Prefixes	9
Lock Prefix	10
Repeat Prefixes	10
REX Prefixes	14
1.3 Opcode	19
1.4 ModRM and SIB Bytes	20
1.5 Displacement Bytes	22
1.6 Immediate Bytes	22
1.7 RIP-Relative Addressing	23
Encoding	23
REX Prefix and RIP-Relative Addressing	24
Address-Size Prefix and RIP-Relative Addressing	24
2 Instruction Details	25
2.1 Instruction Subsets	25
2.2 Reference-Page Format	26
2.3 Summary of Registers and Data Types	28
General-Purpose Instructions	28
128-Bit Media Instructions	31
64-Bit Media Instructions	34
x87 Floating-Point Instructions	36
System Instructions	37
2.4 Summary of Exceptions	39
2.5 Notation	41
Mnemonic Syntax	41
Opcode Syntax	44
Pseudocode Definitions	46

2.6	General-Purpose Instruction Reference	57
	AAA	58
	AAD	59
	AAM	61
	AAS	62
	ADC	63
	ADD	66
	AND	69
	BOUND	72
	BSF	74
	BSR	76
	BSWAP	78
	BT	79
	BTC	81
	BTR	83
	BTS	85
	CALL	87
	CBW	94
	CDQ	95
	CDQE	96
	CLC	97
	CLD	98
	CLFLUSH	99
	CMC	101
	CMOV _{cc}	102
	CMP	106
	CMPS _x	109
	CMPXCHG	112
	CMPXCHG8B	114
	CPUID	116
	CQO	117
	CWD	118
	CWDE	119
	DAA	120
	DAS	121
	DEC	122
	DIV	124
	ENTER	126
	IDIV	128
	IMUL	131
	IN	134
	INC	135
	INS _x	137
	INT	140
	INTO	147
	Jcc	150
	JCXZ	154

JMP	155
LAHF	161
LDS	162
LEA	164
LEAVE	166
LES	167
LFENCE	169
LFS	170
LGS	172
LODS _x	174
LOOP _{cc}	176
LSL	178
LSS	181
MFENCE	183
MOV	184
MOVD	188
MOVMSKPD	191
MOVMSKPS	193
MOVNTI	195
MOVS _x	196
MOVSX	199
MOVSXD	201
MOVZX	203
MUL	205
NEG	207
NOP	209
NOT	210
OR	212
OUT	215
OUTS _x	216
POP	218
POP _{Ax}	221
POP _{Fx}	223
PREFETCH _x	226
PREFETCH _{level}	228
PUSH	230
PUSH _{Ax}	232
PUSH _{Fx}	233
RCL	235
RCR	237
RET	239
ROL	243
ROR	245
SAHF	247
SAL	248
SAR	250
SBB	253

	SCASx	256
	SETcc	259
	SFENCE	261
	SHL	262
	SHLD	264
	SHR	267
	SHRD	269
	STC	272
	STD	273
	STOSx	274
	SUB	276
	TEST	279
	XADD	281
	XCHG	283
	XLATx	285
	XOR	287
2.7	128-Bit Media Instruction Reference	291
	ADDPD	293
	ADDPS	295
	ADDSD	297
	ADDSS	300
	ANDNPD	303
	ANDNPS	305
	ANDPD	307
	ANDPS	309
	CMPPD	311
	CMPPS	314
	CMPSD	317
	CMPS	320
	COMISD	323
	COMISS	326
	CVTDQ2PD	329
	CVTDQ2PS	331
	CVTPD2DQ	333
	CVTPD2PI	335
	CVTPD2PS	338
	CVTPI2PD	341
	CVTPI2PS	343
	CVTPS2DQ	346
	CVTPS2PD	348
	CVTPS2PI	350
	CVTSD2SI	353
	CVTSD2SS	356
	CVTSI2SD	359
	CVTSI2SS	362
	CVTSS2SD	365
	CVTSS2SI	367

CVTTPD2DQ	370
CVTTPD2PI	372
CVTTPS2DQ	375
CVTTPS2PI	377
CVTTSD2SI	380
CVTTSS2SI	383
DIVPD	386
DIVPS	389
DIVSD	392
DIVSS	395
FXRSTOR	398
FXSAVE	400
LDMXCSR	402
MASKMOVDQU	404
MAXPD	406
MAXPS	408
MAXSD	411
MAXSS	414
MINPD	417
MINPS	419
MINSD	422
MINSS	425
MOVAPD	428
MOVAPS	430
MOVD	432
MOVDQ2Q	435
MOVDQA	437
MOVDQU	439
MOVHLPS	441
MOVHPD	443
MOVHPS	445
MOVLHPS	447
MOVLPD	449
MOVLPS	451
MOVMSKPD	453
MOVMSKPS	455
MOVNTDQ	457
MOVNTPD	459
MOVNTPS	461
MOVQ	463
MOVQ2DQ	465
MOVSD	467
MOVSS	469
MOVUPD	471
MOVUPS	473
MULPD	475
MULPS	478

MULSD	481
MULSS	484
ORPD	487
ORPS	489
PACKSSDW	491
PACKSSWB	493
PACKUSWB	495
PADDB	497
PADDD	499
PADDQ	501
PADDSB	503
PADDSW	505
PADDUSB	507
PADDUSW	509
PADDW	511
PAND	513
PANDN	515
PAVGB	517
PAVGW	519
PCMPEQB	521
PCMPEQD	523
PCMPEQW	525
PCMPGTB	527
PCMPGTD	529
PCMPGTW	531
PEXTRW	533
PINSRW	535
PMADDWD	538
PMAXSW	540
PMAXUB	542
PMINSW	544
PMINUB	546
PMOVMSKB	548
PMULHUW	550
PMULHW	552
PMULLW	554
PMULUDQ	556
POR	558
PSADBW	560
PSHUFD	562
PSHUFHW	565
PSHUFLW	568
PSLLD	571
PSLLDQ	573
PSLLQ	575
PSLLW	577
PSRAD	579

PSRAW	581
PSRLD	584
PSRLDQ	586
PSRLQ	588
PSRLW	590
PSUBB	592
PSUBD	594
PSUBQ	596
PSUBSB	598
PSUBSW	600
PSUBUSB	602
PSUBUSW	604
PSUBW	606
PUNPCKHBW	608
PUNPCKHDQ	610
PUNPCKHQDQ	612
PUNPCKHWD	614
PUNPCKLBW	616
PUNPCKLDQ	618
PUNPCKLQDQ	620
PUNPCKLWD	622
PXOR	624
RCPPS	626
RCPSS	628
RSQRTPS	630
RSQRTSS	632
SHUFPD	634
SHUFPS	636
SQRTPD	639
SQRTPS	641
SQRTSD	644
SQRTSS	646
STMXCSR	648
SUBPD	650
SUBPS	653
SUBSD	656
SUBSS	659
UCOMISD	662
UCOMISS	665
UNPCKHPD	668
UNPCKHPS	670
UNPCKLPD	672
UNPCKLPS	674
XORPD	676
XORPS	678
2.8 64-Bit Media Instruction Reference	681
CVTPD2PI	684

CVTPI2PD	687
CVTPI2PS.....	689
CVTPS2PI.....	692
CVTTPD2PI	695
CVTTPS2PI	698
EMMS	701
FEMMS.....	702
FNSAVE	703
FRSTOR	705
FSAVE.....	707
FXRSTOR.....	709
FXSAVE	711
MASKMOVQ	713
MOVD	715
MOVDQ2Q	718
MOVNTQ	720
MOVQ	722
MOVQ2DQ	724
PACKSSDW	726
PACKSSWB	728
PACKUSWB	730
PADDB	732
PADDD	734
PADDQ	736
PADDSB	738
PADDSW.....	740
PADDUSB.....	742
PADDUSW	744
PADDW.....	746
PAND	748
PANDN	750
PAVGB.....	752
PAVGUSB	754
PAVGW	756
PCMPEQB	758
PCMPEQD	760
PCMPEQW.....	762
PCMPGTB	764
PCMPGTD	766
PCMPGTW.....	768
PEXTRW.....	770
PF2ID	772
PF2IW.....	774
PFACC.....	776
PFADD	779
PFCMPEQ	781
PFCMPGE	784

PFCMPGT	787
PFMAX	790
PFMIN	793
PFMUL	796
PFNACC	798
PFPNACC	801
PFRCP	804
PFRCPIT1	807
PFRCPIT2	810
PFRSQIT1	813
PFRSQRT	816
PFSUB	819
PFSUBR	822
PI2FD	825
PI2FW	827
PINSRW	829
PMADDWD	831
PMAXSW	833
PMAXUB	835
PMINSW	837
PMINUB	839
PMOVMSKB	841
PMULHRW	843
PMULHUW	845
PMULHW	847
PMULLW	849
PMULUDQ	851
POR	853
PSADBW	855
PSHUFW	857
PSLLD	860
PSLLQ	862
PSLLW	864
PSRAD	866
PSRAW	868
PSRLD	870
PSRLQ	872
PSRLW	874
PSUBB	876
PSUBD	878
PSUBQ	880
PSUBSB	882
PSUBSW	884
PSUBUSB	886
PSUBUSW	888
PSUBW	890
PSWAPD	892

	PUNPCKHBW	894
	PUNPCKHDQ	896
	PUNPCKHWD	898
	PUNPCKLBW	900
	PUNPCKLDQ	902
	PUNPCKLWD	904
	PXOR	906
2.9	x87 Floating-Point Instruction Reference	909
	F2XM1	910
	FABS	912
	FADDx	914
	FBLD	917
	FBSTP	919
	FCHS	922
	FCLEXx	924
	FCMOV _{cc}	926
	FCOMx	928
	FCOMIx	931
	FCOS	934
	FDECSTP	936
	FDIVx	938
	FDIVRx	941
	FFREE	944
	FICOMx	945
	FILD	948
	FINCSTP	950
	FINITx	952
	FISTx	954
	FLD	957
	FLD1	960
	FLDCW	961
	FLDENV	963
	FLDL2E	965
	FLDL2T	967
	FLDLG2	969
	FLDLN2	971
	FLDPI	973
	FLDZ	975
	FMULx	976
	FNOP	979
	FNSAVE	980
	FPATAN	982
	FPREM	984
	FPREM1	986
	FPTAN	988
	FRNDINT	990
	FRSTOR	992

	FSAVE.....	994
	FSCALE	996
	FSIN	998
	FSINCOS	1000
	FSQRT	1002
	FST	1004
	FSTCW _x	1007
	FSTENV _x	1009
	FSTP	1011
	FSTSW _x	1014
	FSUB _x	1016
	FSUBR _x	1019
	FTST	1022
	FUCOM _x	1024
	FUCOMI _x	1026
	FWAIT	1029
	FXAM	1031
	FXCH	1033
	FXRSTOR	1035
	FXSAVE	1037
	FXTRACT	1039
	FYL2X	1041
	FYL2XP1	1043
	WAIT	1045
2.10	System Instruction Reference	1047
	ARPL	1048
	CLI	1050
	CLTS	1052
	HLT	1053
	INT 3	1054
	INVD	1060
	INVLPG	1061
	IRET _x	1062
	LAR	1068
	LGDT	1071
	LIDT	1073
	LLDT	1075
	LMSW	1077
	LTR	1078
	MOV(CR _n)	1080
	MOV (DR _n)	1082
	RDMSR	1084
	RDPMC	1085
	RDTSR	1086
	RSM	1087
	SGDT	1088
	SIDT	1090

	SLDT	1092
	SMSW	1094
	STI	1096
	STR	1098
	SWAPGS	1100
	SYSCALL	1102
	SYSENTER	1107
	SYSEXIT	1109
	SYSRET	1111
	UD2	1115
	VERR	1116
	VERW	1118
	WBINVD	1120
	WRMSR	1121
3	Opcode and Operand Encodings	1123
3.1	Opcode-Syntax Notation	1123
3.2	Opcode Encodings	1126
	One-Byte Opcodes	1126
	Two-Byte Opcodes	1128
	rFLAGS Condition Codes for Two-Byte Opcodes	1137
	ModRM Extensions to One-Byte and Two-Byte Opcodes	1138
	ModRM Extensions to SWAPGS and CLFLUSH Opcodes	1141
	3DNow!™ Opcodes	1141
	x87 Encodings	1144
	rFLAGS Condition Codes for x87 Opcodes	1153
3.3	Operand Encodings	1153
	ModRM Operand References	1153
	SIB Operand References	1159
Appendix A	General-Purpose Instructions in 64-Bit Mode	1163
A.1	General Rules for 64-Bit Mode	1163
A.2	Operation and Operand Size in 64-Bit Mode	1164
A.3	Invalid and Reassigned Instructions in 64-Bit Mode	1195
A.4	Instructions with 64-Bit Default Operand Size	1197
A.5	Single-Byte INC and DEC Instructions in 64-Bit Mode	1199
A.6	NOP in 64-Bit Mode	1199
A.7	Segment Override Prefixes in 64-Bit Mode	1200
Appendix B	Differences Between Long Mode and Legacy Mode	1201
Appendix C	Instruction Subsets and CPUID Feature Sets	1203
C.1	Instruction Subsets	1203
C.2	CPUID Feature Sets	1205
C.3	Instruction List	1207
Appendix D	Instruction Effects on rFLAGS	1247
	Index	1253

Figures

Figure 1-1. Instruction Byte-Order	1
Figure 1-2. Little-Endian Byte-Order of Instruction Stored in Memory . . .	2
Figure 1-3. Encoding Examples of REX-Prefix R, X, and B Bits.	18
Figure 1-4. ModRM-Byte Format	21
Figure 1-5. SIB-Byte Format	21
Figure 2-1. Format of Instruction-Detail Pages	27
Figure 2-2. General Registers in Legacy and Compatibility Modes.	28
Figure 2-3. General Registers in 64-Bit Mode.	29
Figure 2-4. Segment Registers	30
Figure 2-5. General-Purpose Data Types.	31
Figure 2-6. 128-Bit Media Registers.	32
Figure 2-7. 128-Bit Media Data Types	33
Figure 2-8. 64-Bit Media Registers.	34
Figure 2-9. 64-Bit Media Data Types	35
Figure 2-10.x87 Registers.	36
Figure 2-11.x87 Data Types	37
Figure 2-12.System Registers	38
Figure 2-13.System Data Structures	39
Figure 2-14.Mnemonic Syntax for Typical Two-Operand Instruction.	41
Figure 2-15.Diagram Conventions for 128-Bit Media Instructions	291
Figure 2-16.Diagram Conventions for 64-Bit Media Instructions	681
Figure 3-1. ModRM-Byte <i>reg</i> Field	1138
Figure 3-2. ModRM-Byte Format	1154
Figure 3-3. SIB-Byte Format	1160
Figure C-1. Instruction Subsets vs. CPUID Feature Sets.	1204

Tables

Table 1-1.	Legacy Instruction Prefixes	4
Table 1-2.	Operand-Size Overrides.....	5
Table 1-3.	Address-Size Overrides	7
Table 1-4.	Pointer and Count Registers, and the Address-Size Prefix	8
Table 1-5.	Segment-Override Prefixes.....	9
Table 1-6.	REP Prefix Opcodes.....	11
Table 1-7.	REPE and REPZ Opcodes.....	12
Table 1-8.	REPNE and REPNZ Opcodes	13
Table 1-9.	REX Instruction Prefixes	15
Table 1-10.	Instructions Not Requiring REX Prefix in 64-Bit Mode.....	15
Table 1-11.	REX Prefix-Byte Fields	16
Table 1-12.	Special REX Encodings for Registers	19
Table 1-13.	Encoding for RIP-Relative Addressing	24
Table 2-1.	Interrupt-Vector Source and Cause	40
Table 2-2.	+rb, +rw, +rd, and +rq Register Value.....	45
Table 2-3.	Summary of PREFETCH Instruction Type Options.....	227
Table 2-4.	Locality References for the Prefetch Instructions	228
Table 2-5.	Immediate Operand Values for Compare Operations	312
Table 2-6.	Immediate-Byte Operand Encoding for 128-Bit PEXTRW...	533
Table 2-7.	Immediate-Byte Operand Encoding for 128-Bit PINSRW ...	536
Table 2-8.	Immediate-Byte Operand Encoding for PSHUFD	563
Table 2-9.	Immediate-Byte Operand Encoding for PSHUFHW.....	566
Table 2-10.	Immediate-Byte Operand Encoding for PSHUFLW.....	569
Table 2-11.	Immediate-Byte Operand Encoding for SHUFPD	634
Table 2-12.	Immediate-Byte Operand Encoding for SHUFPS.....	636
Table 2-13.	Immediate-Byte Operand Encoding for 64-Bit PEXTRW...	770
Table 2-14.	Numeric Range for PF2ID Results	772
Table 2-15.	Numeric Range for PF2IW Results	774
Table 2-16.	Numeric Range for PFACC Results	777
Table 2-17.	Numeric Range for the PFADD Results	779
Table 2-18.	Numeric Range for the PFCMPEQ Instruction.....	782
Table 2-19.	Numeric Range for the PFCMPGE Instruction.....	785
Table 2-20.	Numeric Range for the PFCMPGT Instruction.....	788
Table 2-21.	Numeric Range for the PFMAX Instruction	791

Table 2-22. Numeric Range for the PFMIN Instruction	794
Table 2-23. Numeric Range for the PFMUL Instruction	796
Table 2-24. Numeric Range of PFNACC Results	799
Table 2-25. Numeric Range of PFPNACC Result (Low Result)	802
Table 2-26. Numeric Range of PFPNACC Result (High Result)	802
Table 2-27. Numeric Range for the PFRCP Result.	805
Table 2-28. Numeric Range for the PFRCPIT1 Results	808
Table 2-29. Numeric Range for the PFRCPIT2 Results	811
Table 2-30. Numeric Range for the PFRSQIT1 Result.	814
Table 2-31. Numeric Range for the PFRCP Result.	817
Table 2-32. Numeric Range for the PFSUB Results	820
Table 2-33. Numeric Range for the PFSUBR Results	823
Table 2-34. Immediate-Byte Operand Encoding for 64-Bit PINSRW	829
Table 2-35. Immediate-Byte Operand Encoding for PSHUFW	858
Table 3-1. One-Byte Opcodes, Low Nibble 0:7h	1126
Table 3-2. One-Byte Opcodes, Low Nibble 8:Fh	1127
Table 3-3. Second Byte of Two-Byte Opcodes, Low Nibble 0:7h	1129
Table 3-4. Second Byte of Two-Byte Opcodes, Low Nibble 8:Fh	1133
Table 3-5. rFLAGS Condition Codes for CMOVcc, Jcc, and SETcc	1137
Table 3-6. One-Byte and Two-Byte Opcode ModRM Extensions	1139
Table 3-7. SWAPGS and CLFLUSH ModRM Extensions	1141
Table 3-8. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 0:7h	1142
Table 3-9. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 8:Fh	1143
Table 3-10. x87 Opcodes and ModRM Extensions	1145
Table 3-11. rFLAGS Condition Codes for FCMOVcc	1153
Table 3-12. ModRM Register References, 32/64-Bit Address	1155
Table 3-13. ModRM Memory References, 32/64-Bit Address	1156
Table 3-14. ModRM Register References, 16-Bit Address	1157
Table 3-15. ModRM Memory References, 16-Bit Address	1158
Table 3-16. SIB <i>base</i> Field References	1160
Table 3-17. SIB Memory References	1161
Table A-1. Operations and Operands in 64-Bit Mode	1165
Table A-2. Invalid Instructions in 64-Bit Mode	1196
Table A-3. Reassigned Instructions in 64-Bit Mode	1197
Table A-4. Invalid Instructions in Long Mode	1197
Table A-5. Instructions Defaulting to 64-Bit Operand Size	1198

Table B-1. Differences Between Long Mode and Legacy Mode.	1201
Table C-1. Instruction Subsets and CPUID Feature Sets	1207
Table D-1. Instruction Effects on rFLAGS	1247

Preface

Audience

This volume (Volume 3) is intended for all programmers writing application or system software for a processor that implements the x86-64 architecture. Descriptions of general-purpose, 128-bit media, 64-bit media, and x87 floating-point instructions assume an understanding of the application-level programming topics described in Volume 1 of this document. Descriptions of system instructions assume an understanding of the system-level programming topics described in Volume 2 of this document.

Organization

This volume describes the x86-64 architecture's instruction set in detail. It covers their mnemonic syntax, opcodes, functions, affected flags, and possible exceptions. The x86-64 instruction set is divided into five subsets:

- General-Purpose Instructions
- 128-Bit Media Instructions
- 64-Bit Media Instructions
- x87 Floating-Point Instructions
- System Instructions

Several instructions belong to—and are described identically in—multiple instruction subsets.

The index at the end cross-references topics within this volume. For other topics relating to the x86-64 architecture, see the tables of contents and indexes of Volumes 1 and 2.

Definitions

Many of the following definitions assume an in-depth knowledge of the legacy x86 architecture. See “Related Documents” on page xxxiii for descriptions of the legacy x86 architecture.

Terms and Notation

In addition to the notation described below, Section 3.1, “Opcode-Syntax Notation”, on page 1123 describes notation relating specifically to opcodes.

seg:[rSI]

The contents of a memory location whose address is in segment *seg* (for example, GS) and whose relative address is in the rSI register.

1011b

A binary value, in this example a 4-bit value.

F0EAh

A hexadecimal value, in this example a 2-byte value.

[1,2)

A range that includes the left-most value (in this case, 1) but excludes the right-most value (in this case, 2).

7:4

A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first.

128-bit media instructions

Instructions that uses the 128-bit XMM registers. These are a combination of SSE and SSE-2 instruction sets.

64-bit media instructions

Instructions that uses the 64-bit MMX™ registers. These are primarily a combination of MMX and 3DNow!™ instruction sets, with some additional instructions from the SSE and SSE-2 instruction sets.

16-bit mode

Legacy mode or compatibility mode in which a 16-bit address size is active. See *legacy mode* and *compatibility mode*.

32-bit mode

Legacy mode or compatibility mode in which a 32-bit address size is active. See *legacy mode* and *compatibility mode*.

64-bit mode

A submode of *long mode*. In 64-bit mode, the default address size is 64 bits and new features, such as register extensions, are supported for system and application software.

#GP(0)

Notation indicating a general-protection exception (#GP) with error code of 0.

absolute

Said of a displacement that references the base of a code segment rather than an instruction pointer. Contrast with *relative*.

biased exponent

The sum of a floating-point value's exponent and a constant bias for a particular floating-point data type. The bias makes the range of the biased exponent always positive, which allows reciprocation without overflow.

byte

Eight bits.

clear

To write a bit-value of 0. Compare *set*.

compatibility mode

A submode of *long mode*. In compatibility mode, the default address size is 32 bits and legacy 16-bit and 32-bit applications run without modification.

b

A bit, as in *1Mb* for one megabit, or *lsb* for least-significant bit.

B

A byte, as in *1MB* for one megabyte, or *LSB* for least-significant byte.

commit

To irreversibly write, in program order, an instruction's result to software-visible storage, such as a register (including flags), the data cache, an internal write buffer, or memory.

CPL

Current privilege level.

CR0–CR4

A register range, from register CR0 through CR4, inclusive, with the low-order register first.

CR0.PE=1

Notation indicating that the PE bit of the CR0 register has a value of 1.

direct

Referencing a memory location whose address is included in the instruction's syntax as an immediate operand. The address may be an absolute or relative address. Compare *indirect*.

dirty data

Data held in the processor's caches or internal buffers that is more recent than the copy held in main memory.

displacement

A signed value that is added to the base of a segment (absolute addressing) or an instruction pointer (relative addressing). Same as *offset*.

doubleword

Two words, or four bytes, or 32 bits.

double quadword

Eight words, or 16 bytes, or 128 bits. Also called *octword*.

DS:rSI

The contents of a memory location whose segment is DS and whose byte address is located in the rSI register.

EFER.LME=0

Notation indicating that the LME bit of the EFER register has a value of 0.

effective address size

The address size for the current instruction after accounting for the default address size and any address-size override prefix.

effective operand size

The operand size for the current instruction after accounting for the default operand size and any operand-size override prefix.

element

See *vector*.

exception

An abnormal condition that occurs as the result of executing an instruction. The processor's response to an exception depends on the type of the exception. For all exceptions except 128-bit media SIMD floating-point exceptions and x87 floating-point exceptions, control is transferred to the handler (or service routine) for that exception, as defined by the exception's vector. For floating-point exceptions defined by the IEEE 754 standard, there are both masked and unmasked responses. When unmasked, the exception handler is called, and when masked a default response is provided instead of calling the handler.

FF /0

Notation indicating that FF is the first byte of an opcode, and a sub-field in the second byte has a value of 0.

flush

An often ambiguous term meaning (1) writeback, if modified, and invalidate, as in “flush the cache line”, or (2) invalidate, as in “flush the pipeline”, or (3) change a value, as in “flush to zero”.

GDT

Global descriptor table.

IDT

Interrupt descriptor table.

IGN

Ignore. Field is ignored.

indirect

Referencing a memory location whose address is in a register or other memory location. The address may be an absolute or relative address. Compare *direct*.

IRB

The virtual-8086 mode interrupt-redirection bitmap.

IST

The long-mode interrupt-stack table.

IVT

The real-address mode interrupt-vector table.

LDT

Local descriptor table.

legacy x86

The legacy x86 architecture. See “Related Documents” on page xxxiii for descriptions of the legacy x86 architecture.

legacy mode

An operating mode of the x86-64 architecture in which existing 16-bit and 32-bit applications and operating systems run without modification. A processor implementation of the x86-64 architecture can run in either *long mode* or *legacy mode*. Legacy mode has three submodes, *real mode*, *protected mode*, and *virtual-8086 mode*.

long mode

An operating mode unique to the x86-64 architecture. A processor implementation of the x86-64 architecture can run in either *long mode* or *legacy mode*. Long mode has two submodes, *64-bit mode* and *compatibility mode*.

lsb

least-significant bit.

LSB

least-significant byte.

main memory

Physical memory, such as RAM and ROM (but not cache memory) that is installed in a particular computer system.

mask

(1) A control bit that prevents the occurrence of a floating-point exception from invoking an exception handling routine. (2) A field of bits used for a control purpose.

MBZ

Must be zero. If software attempts to set an MBZ bit to 1, a general-protection exception (#GP) occurs.

memory

Unless otherwise specified, *main memory*.

ModRM

A byte following an instruction opcode that specifies address calculation based on mode (mod), register (r), and memory (m) variables.

moffset

A direct memory offset. I.e., a displacement that is added to the base of a code segment (for absolute addressing) or to an instruction pointer (for addressing relative to the instruction pointer, as in RIP-relative addressing).

msb

Most-significant bit.

MSB

Most-significant byte.

multimedia instructions

A combination of 128-bit *media instructions* and 64-bit *media instructions*.

octword

Same as *double quadword*.

offset

Same as *displacement*.

overflow

The condition in which a floating-point number is larger in magnitude than the largest, finite, positive or negative number that can be represented in the data-type format being used.

packed

See *vector*.

PAE

Physical-address extensions.

physical memory

Actual memory, consisting of *main memory* and cache.

probe

A check for an address in a processor's caches or internal buffers. *External probes* originate outside the processor, and *internal probes* originate within the processor.

protected mode

A submode of *legacy mode*.

quadword

Four words, or eight bytes, or 64 bits.

RAZ

Read as zero (0), regardless of what it written.

real-address mode

See *real mode*.

real mode

A short name for *real-address mode*, a submode of *legacy mode*.

relative

Referencing with a displacement (also called offset) from an instruction pointer rather than the base of a code segment. Contrast with *absolute*.

REX

An instruction prefix that specifies a 64-bit operand size and provides access to additional registers.

RIP-Relative Addressing

Addressing relative to the 64-bit RIP instruction pointer. Compare *offset*.

set

To write a bit-value of 1. Compare *clear*.

SIB

A byte following an instruction opcode that specifies address calculation based on scale (S), index (I), and base (B).

SIMD

Single instruction, multiple data. See *vector*.

SSE

Streaming SIMD extensions instruction set. See *128-bit media instructions* and *64-bit media instructions*.

SSE-2

Extensions to the SSE instruction set. See *128-bit media instructions* and *64-bit media instructions*.

sticky bit

A bit that set or cleared by hardware and that remains in that state until explicitly changed by software.

TOP

x87 top-of-stack pointer.

TPR

Task-priority register (CR8).

TSS

Task state segment.

underflow

The condition in which a floating-point number is smaller in magnitude than the smallest non-zero, positive or negative number that can be represented in the data-type format being used.

vector

(1) A set of integer or floating-point values, called *elements*, that are packed into a single operand. Most of the 128-bit and 64-bit media instructions use vectors as operands. Vectors are also called *packed* or *SIMD* (single-instruction multiple-data) operands.

(2) An index into an interrupt descriptor table (IDT), used to for accessing exception handlers. Compare *exception*.

virtual-8086 mode

A submode of *legacy mode*.

word

Two bytes, or 16 bits.

x86

See *legacy x86*.

Registers

In the following list of registers, the names are used to refer either to the register or to the contents of the that register:

AH–DH

The high 8-bit AH, BH, CH, and DH registers. Compare *AL–DL*.

AL–DL

The low 8-bit AL, BL, CL, and DL registers. Compare *AH–DH*.

AL–r15B

The low 8-bit AL, BL, CL, DL, SIL, DIL, BPL, SPL, and R8B–R15B registers, available in 64-bit mode.

eAX–eSP

The 16-bit AX, BX, CX, DX, DI, SI, BP, SP registers or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP registers. Compare *rAX–rSP*.

EBP

Extended base pointer register.

BP

Base pointer register.

CR_n

Control register number *n*.

CS

Code segment register.

EFER

Extended features enable register.

eFLAGS

16-bit or 32-bit flags register. Compare *rFLAGS*.

EFLAGS

32-bit (extended) flags register.

eIP

16-bit or 32-bit instruction-pointer register. Compare *rIP*.

EIP

32-bit (extended) instruction-pointer register.

FLAGS

16-bit flags register.

GDTR

Global descriptor table register.

GPRs

General-purpose registers. For the 16-bit data size, these are AX, BX, CX, DX, DI, SI, BP, SP. For the 32-bit data size, these are EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP. For the 64-bit data size, these include RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, and R8–R15.

IDTR

Interrupt descriptor table register.

IP

16-bit instruction-pointer register.

LDTR

Local descriptor table register.

MSR

Model-specific register.

r8–r15

The 8-bit R8B–R15B registers, or the 16-bit R8W–R15W registers, or the 32-bit R8D–R15D registers, or the 64-bit R8–R15 registers.

rAX–rSP

The 16-bit AX, BX, CX, DX, DI, SI, BP, SP registers, or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP registers, or the 64-bit RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP registers. The “r” variable should be replaced by nothing for 16-bit size, “E” for 32-bit size, or “R” for 64-bit size.

RAX

64-bit version of EAX register.

RBP

64-bit version of EBP register.

RBX

64-bit version of EBX register.

RCX

64-bit version of ECX register.

RDI

64-bit version of EDI register.

RDX

64-bit version of EDX register.

rFLAGS

16-bit, 32-bit, or 64-bit flags register. Compare *RFLAGS*.

RFLAGS

64-bit flags register. Compare *rFLAGS*.

rIP

16-bit, 32-bit, or 64-bit instruction-pointer register. Compare *RIP*.

RIP

64-bit instruction-pointer register.

RSI

64-bit version of ESI register.

RSP

64-bit version of ESP register.

SP

Stack pointer register.

SS

Stack segment register.

TPR

Task priority register, a new register introduced in the x86-64 architecture to speed interrupt management.

TR

Task register.

Endian Order

The x86 and x86-64 architectures address memory using little-endian byte-ordering. Multi-byte values are stored with their

least-significant byte (LSB) at the lowest byte address, and they are illustrated with their least significant byte at the right side. Strings are illustrated in reverse order, because the addresses of their bytes increase from right to left.

Related Documents

- Peter Abel, *IBM PC Assembly Language and Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Rakesh Agarwal, *80x86 Architecture & Programming: Volume II*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- AMD, *AMD-K6™ MMX™ Enhanced Processor Multimedia Technology*, Sunnyvale, CA, 2000.
- AMD, *3DNow!™ Technology Manual*, Sunnyvale, CA, 2000.
- AMD, *AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets*, Sunnyvale, CA, 2000.
- Don Anderson and Tom Shanley, *Pentium Processor System Architecture*, Addison-Wesley, New York, 1995.
- Nabajyoti Barkakati and Randall Hyde, *Microsoft Macro Assembler Bible*, Sams, Carmel, Indiana, 1992.
- Barry B. Brey, *8086/8088, 80286, 80386, and 80486 Assembly Language Programming*, Macmillan Publishing Co., New York, 1994.
- Barry B. Brey, *Programming the 80286, 80386, 80486, and Pentium Based Personal Computer*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Ralf Brown and Jim Kyle, *PC Interrupts*, Addison-Wesley, New York, 1994.
- Penn Brumm and Don Brumm, *80386/80486 Assembly Language Programming*, Windcrest McGraw-Hill, 1993.
- Geoff Chappell, *DOS Internals*, Addison-Wesley, New York, 1994.
- Chips and Technologies, Inc. *Super386 DX Programmer's Reference Manual*, Chips and Technologies, Inc., San Jose, 1992.
- John Crawford and Patrick Gelsinger, *Programming the 80386*, Sybex, San Francisco, 1987.
- Cyrix Corporation, *5x86 Processor BIOS Writer's Guide*, Cyrix Corporation, Richardson, TX, 1995.

- Cyrix Corporation, *M1 Processor Data Book*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor MMX Extension Opcode Table*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor Data Book*, Cyrix Corporation, Richardson, TX, 1997.
- Jeffrey P. Doyer, *Introduction to Protected Mode Programming*, course materials for an onsite class, 1992.
- Ray Duncan, *Extending DOS: A Programmer's Guide to Protected-Mode DOS*, Addison Wesley, NY, 1991.
- William B. Giles, *Assembly Language Programming for the Intel 80xxx Family*, Macmillan, New York, 1991.
- Frank van Gilluwe, *The Undocumented PC*, Addison-Wesley, New York, 1994.
- John L. Hennessy and David A. Patterson, *Computer Architecture*, Morgan Kaufmann Publishers, San Mateo, CA, 1996.
- Thom Hogan, *The Programmer's PC Sourcebook*, Microsoft Press, Redmond, WA, 1991.
- Hal Katircioglu, *Inside the 486, Pentium, and Pentium Pro*, Peer-to-Peer Communications, Menlo Park, CA, 1997.
- IBM Corporation, *486SLC Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *486SLC2 Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *80486DX2 Processor Floating Point Instructions*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *80486DX2 Processor BIOS Writer's Guide*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *Blue Lightning 486DX2 Data Book*, IBM Corporation, Essex Junction, VT, 1994.
- Institute of Electrical and Electronics Engineers, *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985.
- Institute of Electrical and Electronics Engineers, *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, ANSI/IEEE Std 854-1987.

- Muhammad Ali Mazidi and Janice Gillispie Mazidi, *80X86 IBM PC and Compatible Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- Hans-Peter Messmer, *The Indispensable Pentium Book*, Addison-Wesley, New York, 1995.
- Karen Miller, *An Assembly Language Introduction to Computer Architecture: Using the Intel Pentium*, Oxford University Press, New York, 1999.
- Stephen Morse, Eric Isaacson, and Douglas Albert, *The 80386/387 Architecture*, John Wiley & Sons, New York, 1987.
- NexGen Inc., *Nx586 Processor Data Book*, NexGen Inc., Milpitas, CA, 1993.
- NexGen Inc., *Nx686 Processor Data Book*, NexGen Inc., Milpitas, CA, 1994.
- Bipin Patwardhan, *Introduction to the Streaming SIMD Extensions in the Pentium III*, www.x86.org/articles/sse_pt1/simd1.htm, June, 2000.
- Peter Norton, Peter Aitken, and Richard Wilton, *PC Programmer's Bible*, Microsoft Press, Redmond, WA, 1993.
- *PharLap 386/ASM Reference Manual*, Pharlap, Cambridge MA, 1993.
- *PharLap TNT DOS-Extender Reference Manual*, Pharlap, Cambridge MA, 1995.
- Sen-Cuo Ro and Sheau-Chuen Her, *i386/i486 Advanced Programming*, Van Nostrand Reinhold, New York, 1993.
- Tom Shanley, *Protected Mode System Architecture*, Addison Wesley, NY, 1996.
- SGS-Thomson Corporation, *80486DX Processor SMM Programming Manual*, SGS-Thomson Corporation, 1995.
- Walter A. Triebel, *The 80386DX Microprocessor*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- John Wharton, *The Complete x86*, MicroDesign Resources, Sebastopol, California, 1994.
- Web sites:
 - www.amd.com
 - news.comp.arch
 - news.comp.lang.asm.x86
 - news.intel.microprocessors

- news.microsoft

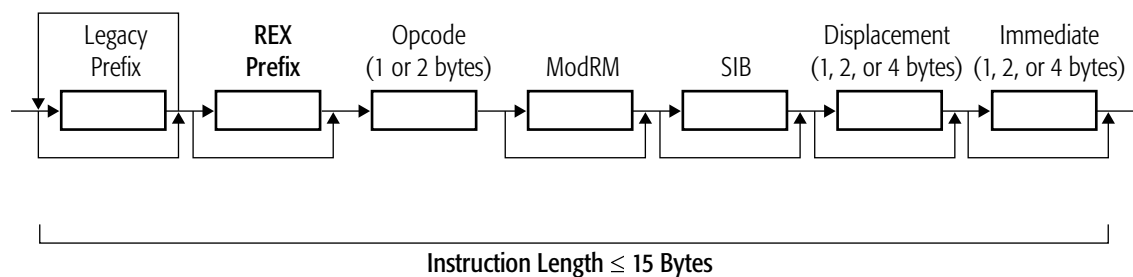
1 Instruction Formats

An instruction's format encodes its operation, and it can also encode the locations of operands and result. This section describes the general format and parameters used by all instructions. The specific format(s) for each instruction are given in the following five sections of this volume:

- Section 2.6, “General-Purpose Instruction Reference”, on page 45.
- Section 2.7, “128-Bit Media Instruction Reference”, on page 415.
- Section 2.8, “64-Bit Media Instruction Reference”, on page 787.
- Section 2.9, “x87 Floating-Point Instruction Reference”, on page 1017.
- Section 2.10, “System Instruction Reference”, on page 1229.

1.1 Instruction Byte Order

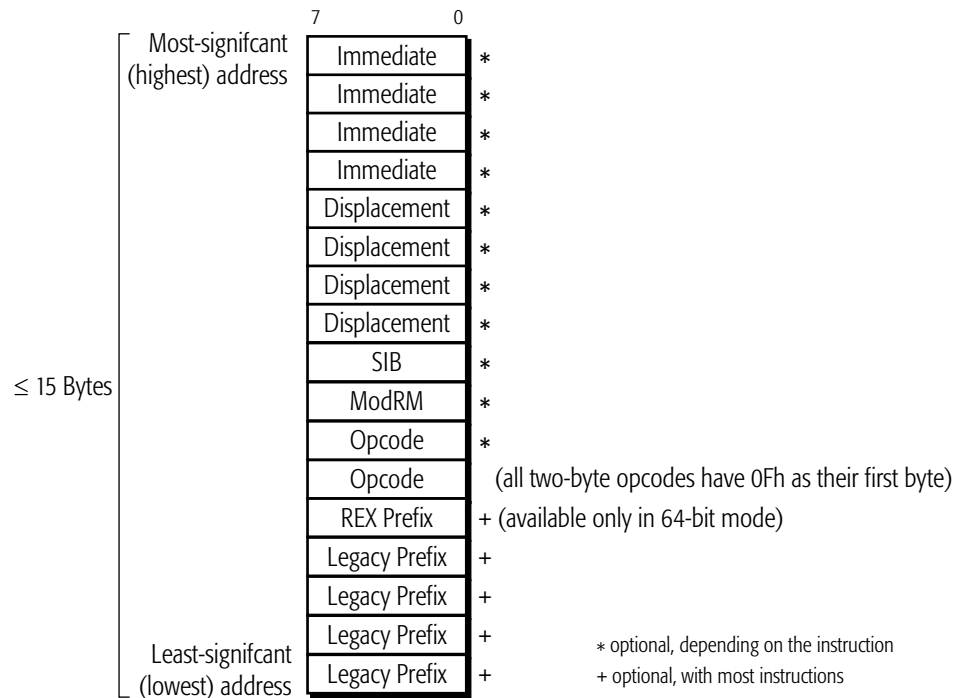
An instruction can be between 1 and 15 bytes in length. Figure 1-1 shows the byte order of the instruction format.



513-303.eps

Figure 1-1. Instruction Byte-Order

Instructions are stored in memory in little-endian order. The least-significant byte of an instruction is stored at its lowest memory address, as shown in Figure 1-2.



513-304.eps

Figure 1-2. Little-Endian Byte-Order of Instruction Stored in Memory

All instructions have an *opcode* that specifies the instruction's basic operation. The opcode is 1 or 2 bytes long, as described in Section 1.3 on page 19. An opcode can be preceded by up to four *legacy prefixes*—one from each of four groups of prefixes, as described in Section 1.2 on page 3. The legacy prefixes modify an instruction's default address size, operand size, or segment, or they invoke a special function such as modification of the opcode, atomic bus-locking, or repetition. The *REX prefix* can be used in 64-bit mode to access the register extensions illustrated in Figure 1-1 on page 2 of Volume 1. If a REX prefix is used, it must immediately precede the first opcode byte.

An instruction's opcode consists of one or two bytes. In several 128-bit and 64-bit media instructions, a legacy operand-size or repeat prefix byte is used in a special-purpose way to modify the opcode. The opcode can be followed by a *mode-register-memory (ModRM) byte*, which further describes the operation and/or operands. The opcode, or the opcode and ModRM byte, can also be followed by a *scale-index-base (SIB) byte*, which

describes the scale, index, and base forms of memory addressing. The ModRM and SIB bytes are described in Section 1.4 on page 20, but their legacy functions can be modified by the REX prefix (Section 1.2 on page 3).

The 15-byte instruction-length limit can only be exceeded by using non-redundant prefixes. If the limit is exceeded, a general-protection exception occurs.

1.2 Instruction Prefixes

The instruction prefixes shown in Figure 1-1 are of two types: legacy prefixes and REX prefixes. Each of the legacy prefixes has a unique byte value. By contrast, the REX prefixes, which enable use of the x86-64 register extensions in 64-bit mode, are organized as a group of byte values in which the value of the prefix indicates the combination of register-extension features to be enabled.

1.2.1 Summary of Legacy Prefixes

Table 1-1 shows the legacy prefixes—i.e., all prefixes except the REX prefixes, which are described on page 14. The legacy prefixes are organized into four groups, as shown in the left-most column of Table 1-1. A maximum of one prefix from each of the four groups can be used in a single instruction. The legacy prefixes can appear in any order within the position shown in Figure 1-1 for legacy prefixes. The result of using multiple prefixes from a single group is unpredictable.

Some of the restrictions on legacy prefixes are:

- *Operand-Size Override*—This prefix affects only general-purpose instructions and a few x87 instructions. When used with 128-bit and 64-bit media instructions, this prefix acts in a special way to modify the opcode.
- *Address-Size Override*—This prefix affects only memory operands.
- *Segment Override*—In 64-bit mode, the CS, DS, ES, and SS segment override prefixes are ignored.
- *LOCK Prefix*—This prefix affects only certain instructions that reference memory.
- *Repeat Prefixes*—These prefixes affect only certain string instructions. When used with 128-bit and 64-bit media instructions, two of the three prefixes act in a special way to modify the opcode.

Table 1-1. Legacy Instruction Prefixes

Prefix Group ⁵	Mnemonic	Prefix Byte (Hex)	Description
Operand-Size Override	none	66 ¹	Changes the default operand size of a memory or register operand, as shown in Table 1-2 on page 5.
Address-Size Override	none	67 ²	Changes the default address size of a memory operand, as shown in Table 1-3 on page 7.
Segment Override	CS	2E ⁶	Forces use of the current CS segment for memory operands.
	DS	3E ⁶	Forces use of the current DS segment for memory operands.
	ES	26 ⁶	Forces use of the current ES segment for memory operands.
	FS	64	Forces use of the current FS segment for memory operands.
	GS	65	Forces use of the current GS segment for memory operands.
	SS	36 ⁶	Forces use of the current SS segment for memory operands.
Lock or Repeat	LOCK	F0 ⁴	Causes certain kinds of memory read-modify-write instructions to occur atomically.
	REP	F3 ³	Repeats a string operation (INS, MOVSB, OUTSB, LODSB, and STOSB) until the rCX register equals 0.
	REPE or REPZ		Repeats a compare-string or scan-string operation (CMPSB and SCASB) until the rCX register equals 0 or the zero flag (ZF) is cleared to 0.
	REPNE or REPNZ	F2 ³	Repeats a compare-string or scan-string operation (CMPSB and SCASB) until the rCX register equals 0 or the zero flag (ZF) is set to 1.

Note:

1. When used with 128-bit and 64-bit media instructions, this prefix acts in a special way to modify the opcode. The prefix is ignored by 64-bit media floating-point (3DNow!) instructions. See "Instructions that Cannot Use the Operand-Size Prefix" on page 6.
2. This prefix changes address size only for a memory operand, and only a single memory operand can be overridden in a given instruction. The prefix is ignored by instructions that have no memory operand.
3. This prefix should be used only with compare-string and scan-string instructions. When used with 128-bit and 64-bit media instructions, the prefix acts in a special way to modify the opcode.
4. The LOCK prefix should not be used for instructions other than those listed in Section 1.2.5, "Lock Prefix", on page 10.
5. A maximum of one prefix from each of the four groups can be used with a single instruction.
6. In 64-bit mode, the CS, DS, ES, and SS segment overrides are ignored.

1.2.2 Operand-Size Override Prefix

The default operand size for an instruction is determined by a combination of its opcode, the D (default) bit in the current code-segment descriptor, and the current operating mode, as shown in Table 1-2. The operand-size override prefix (66h) selects the non-default operand size. The prefix can be used with any general-purpose instruction that accesses non-fixed-size operands in memory or general-purpose registers (GPRs), and it can also be used with the x87 FLDENV, FNSTENV, FNSAVE, and FRSTOR instructions.

In 64-bit mode, the prefix allows mixing of 16-bit, 32-bit, and 64-bit data on an instruction-by-instruction basis. In compatibility and legacy modes, the prefix allows mixing of 16-bit and 32-bit on an instruction-by-instruction basis.

Table 1-2. Operand-Size Overrides

Operating Mode		Default Operand Size (Bits)	Effective Operand Size (Bits)	Instruction Prefix ¹	
				66h	REX ³
Long Mode	64-Bit Mode	32 ²	64	don't care	yes
			32	no	no
			16	yes	no
	Compatibility Mode	32	32	no	Not Applicable
			16	yes	
		16	32	yes	
			16	no	
	Legacy Mode (Protected, Virtual-8086, or Real Mode)	32	32	no	
16			yes		
16		32	yes		
		16	no		

Note:

- “no” indicates that the default operand size is used.
- This is the typical default, although some instructions default to other operand sizes. See Appendix A on page 1297 for details.
- See Section 1.2.7, “REX Prefixes”, on page 14.

In 64-bit mode, most instructions default to a 32-bit operand size. For these instructions, a REX prefix (page 15) specifies a 64-bit operand size, and a 66h prefix specifies a 16-bit operand size. The REX prefix takes precedence over the 66h prefix. However, if an instruction defaults to a 64-bit operand size, it does not need a REX prefix and it can only be overridden to a 16-bit operand size. It cannot be overridden to a 32-bit operand size, because there is no 32-bit operand-size override prefix in 64-bit mode. Two groups of instructions have a default a 64-bit operand size in 64-bit mode:

- Near branches. See Section , “Near Branches in 64-Bit Mode”, on page 99 of Volume 1 for details.
- All instructions, except far branches, that implicitly reference the RSP. See Section 2.3.2, “Stack Pointers”, on page 23 of Volume 1 for details.

Instructions that Cannot Use the Operand-Size Prefix. The operand-size prefix should be used only with general-purpose instructions and the x87 FLDENV, FNSTENV, FNSAVE, and FRSTOR instructions, in which the prefix selects between 16-bit and 32-bit operand size. The prefix is ignored by all other x87 instructions and by 64-bit media floating-point (3DNow!) instructions.

When used with 64-bit media *integer* instructions, the 66h prefix acts in a special way to modify the opcode. This modification typically causes an access to an XMM register or 128-bit memory operand and thereby converts the 64-bit media instruction into its comparable 128-bit media instruction. The result of using an F2h or F3h repeat prefix along with a 66h prefix in 128-bit or 64-bit media instructions is unpredictable.

Operand-Size and REX Prefixes. For non-byte operations, the REX operand-size prefix takes precedence over the 66h prefix. See “REX.W: Operand Width” on page 16 for details.

1.2.3 Address-Size Override Prefix

The default address size for instructions that access non-stack memory is determined by the current operating mode, as shown in Table 1-3. The address-size override prefix (67h) selects the non-default address size. Depending on the operating mode, this prefix allows mixing of 16-bit and 32-bit, or of 32-bit and 64-bit addresses, on an instruction-by-instruction basis. The prefix changes address size only for memory operands and is thus meaningful only with instructions that accesses operands in

memory. It is ignored by instructions that have no memory operands.

For instructions that access a stack segment (SS), the default address size for stack accesses is determined by the D (default) bit in the stack-segment descriptor. In 64-bit mode, the D bit is ignored and all stack references have a 64-bit address size. However, if a instruction accesses both stack and non-stack memory, the address size of the non-stack access is determined in the non-stack way described above.

Table 1-3. Address-Size Overrides

Operating Mode		Default Address Size (Bits)	Effective Address Size (Bits)	Address-Size Prefix (67h) ¹ Required?
Long Mode	64-Bit Mode	64	64	no
			32	yes
	Compatibility Mode	32	32	no
			16	yes
		16	32	yes
			16	no
Legacy Mode (Protected, Virtual-8086, or Real Mode)		32	32	no
			16	yes
		16	32	yes
			16	no

Note:

1. “no” indicates that the default address size is used.

As Table 1-3 shows, the default address size is 64 bits in 64-bit mode. The size can be overridden to 32 bits, but 16-bit addresses are not supported in 64-bit mode. In compatibility and legacy modes, the default address size is 16 bits or 32 bits, depending on the operating mode (see Section 14, “Processor Initialization and Long-Mode Activation”, on page 359 of Volume 2 for details). In these modes, the address-size prefix selects the non-default size, but the 64-bit address size is not available.

Certain instructions reference pointer registers or count registers implicitly, rather than explicitly. In such instructions, the address-size prefix affects the size of such addressing and count registers, just it does when such registers are explicitly referenced. Table 1-3 lists all such instructions and the registers referenced using the three possible address sizes.

Table 1-4. Pointer and Count Registers, and the Address-Size Prefix

Instruction	Pointer or Count Register		
	16-Bit Address Size	32-Bit Address Size	64-Bit Address Size
CMPS, CMPSB, CMPSW, CMPSD, CMPSQ —Compare Strings	SI, DI	ESI, EDI	RSI, RDI
INS, INSB, INSD, INSW —Input String	DI	EDI	RDI
JCXZ, JECXZ, JRCXZ —Jump on CX/ECX/RCX Zero	CX	ECX	RCX
LODS, LODSB, LODSW, LODSD, LODSQ —Load String	SI	ESI	RSI
LOOP, LOOPE, LOOPNZ, LOOPNE, LOOPZ —Loop	CX	ECX	RCX
MOVS, MOVSB, MOVSW, MOVSD, MOVSQ —Move String	SI, DI	ESI, EDI	RSI, RDI
OUTS, OUTSB, OUTSD, OUTSW —Output String	SI	ESI	RSI
REP, REPE, REPNE, REPNZ, REPZ —Repeat Prefixes	CX	ECX	RCX
SCAS, SCASB, SCASW, SCASD, SCASQ —Scan String	DI	EDI	RDI
STOS, STOSB, STOSW, STOSD, STOSQ —Store String	DI	EDI	RDI
XLAT, XLATB —Table Look-up Translation	BX	EBX	RBX

1.2.4 Segment-Override Prefixes

Segment overrides can be used only with instructions that reference non-stack memory. Most instructions that reference memory are encoded with a ModRM byte (page 20). The default segment for such memory-referencing instructions is implied by the base register indicated in its ModRM byte, as follows:

- *Instructions that Reference a Non-Stack Segment*—If an instruction's encoding references any base register other than rBP or rSP, or if an instruction contains an immediate offset, the default segment is the data segment (DS). These instructions can use the segment-override prefix to select one of the non-default segments, as shown in Table 1-5. See the paragraphs below for details when operating in 64-bit mode.
- *String Instructions*—String instructions reference two memory operands. By default, they reference both the DS and ES segments (DS:rSI and ES:rDI). In legacy and compatibility modes, these instructions can override their DS-segment reference, as shown in Table 1-5, but they cannot override their ES-segment reference. See the paragraphs below for details when operating in 64-bit mode.
- *Instructions that Reference the Stack Segment*—If an instruction's encoding references the rBP or rSP base register, the default segment is the stack segment (SS). All instructions that reference the stack (push, pop, call, interrupt, return from interrupt) use the SS segment by default. These instructions cannot use the segment-override prefix.

Table 1-5. Segment-Override Prefixes

Mnemonic	Prefix Byte (Hex)	Description
CS ¹	2E	Forces use of current CS segment for memory operands.
DS ¹	3E	Forces use of current DS segment for memory operands.
ES ¹	26	Forces use of current ES segment for memory operands.
FS	64	Forces use of current FS segment for memory operands.
Note: 1. In 64-bit mode, the CS, DS, ES, and SS segment overrides are ignored.		

Table 1-5. Segment-Override Prefixes (continued)

Mnemonic	Prefix Byte (Hex)	Description
GS	65	Forces use of current GS segment for memory operands.
SS ¹	36	Forces use of current SS segment for memory operands.
Note: 1. In 64-bit mode, the CS, DS, ES, and SS segment overrides are ignored.		

Segment Overrides in 64-Bit Mode. In 64-bit mode, the CS, DS, ES, and SS segment-override prefixes have no effect. These four prefixes are not treated as segment-override prefixes for the purposes of the multiple-prefix (maximum of four) rule. Instead, they are treated as null prefixes.

The FS and GS segment-override prefixes are treated as true segment-override prefixes in 64-bit mode. Use of the FS or GS prefix causes their respective segment bases to be added to the effective address calculation. See “FS and GS Selectors in 64-Bit Mode” on page 103 of Volume 2 for details.

1.2.5 Lock Prefix

The LOCK prefix causes certain kinds of memory read-modify-write instructions to occur atomically. The mechanism for doing so is implementation-dependent (for example, the mechanism may involve bus signaling or packet messaging between the processor and a memory controller). The prefix is intended to give the processor exclusive use of shared memory in a multiprocessor system.

The prefix can only be used with forms of the following instructions that write a memory operand: ADC, ADD, AND, BTC, BTR, BTS, CMPXCHG, CMPXCHG8B, DEC, INC, NEG, NOT, OR, SBB, SUB, XADD, XCHG, and XOR. An invalid-opcode exception occurs if the LOCK prefix is used with any other instruction.

1.2.6 Repeat Prefixes

The repeat prefixes cause repetition of certain instructions that load, store, move, input, or output strings. The prefixes should only be used with such string instructions. Two pairs of repeat prefixes, REPE/REPZ and REPNE/REPZ, perform the same repeat functions for certain compare-string and scan-string instructions. The repeat function uses rCX as a count register. The operand size of rCX is based on address size, as shown in Table 1-4 on page 8.

REP. The REP prefix repeats its associated string instruction the number of times specified in the counter register (rCX). It terminates the repetition when the value in rCX reaches 0. The prefix can only be used with the INS, LODS, MOVS, OUTS, and STOS instructions. Table 1-6 shows the valid REP prefix opcodes.

Table 1-6. REP Prefix Opcodes

Mnemonic	Opcode
REP INS <i>reg/mem8</i> , DX REP INSB	F3 6C
REP INS <i>reg/mem16/32</i> , DX REP INSW REP INSD	F3 6D
REP LODS <i>mem8</i> REP LODSB	F3 AC
REP LODS <i>mem16/32/64</i> REP LODSW REP LODSD REP LODSQ	F3 AD
REP MOVS <i>mem8, mem8</i> REP MOVSB	F3 A4
REP MOVS <i>mem16/32/64, mem16/32/64</i> REP MOVSW REP MOVSD REP MOVSQ	F3 A5
REP OUTS DX, <i>reg/mem8</i> REP OUTSB	F3 6E

Table 1-6. REP Prefix Opcodes (continued)

Mnemonic	Opcode
REP OUTS DX, <i>reg/mem16/32</i> REP OUTSW REP OUTSD	F3 6F
REP STOS <i>mem8</i> REP STOSB	F3 AA
REP STOS <i>mem16/32/64</i> REP STOSW REP STOSD REP STOSQ	F3 AB

REPE and REPZ. REPE and REPZ are synonyms and have identical opcodes. These prefixes repeat their associated string instruction the number of times specified in the counter register (rCX). The repetition terminates when the value in rCX reaches 0 or when the zero flag (ZF) is cleared to 0. The prefix can only be used with the CMPS, CMPSB, CMPSD, CMPSW, SCAS, SCASB, SCASD, and SCASW instructions. Table 1-7 shows the valid REPE and REPZ prefix opcodes.

Table 1-7. REPE and REPZ Opcodes

Mnemonic	Opcode
REPx CMPS <i>mem8, mem8</i> REPx CMPSB	F3 A6
REPx CMPS <i>mem16/32/64, mem16/32/64</i> REPx CMPSW REPx CMPSD REPx CMPSQ	F3 A7
REPx SCAS <i>mem8</i> REPx SCASB	F3 AE
REPx SCAS <i>mem16/32/64</i> REPx SCASW REPx SCASD REPx SCASQ	F3 AF

REPNE and REPNZ. REPNE and REPNZ are synonyms and have identical opcodes. These prefixes repeat their associated string instruction the number of times specified in the counter register (rCX). The repetition terminates when the value in rCX reaches 0 or when the zero flag (ZF) is set to 1. The prefix can only be used with the CMPS, CMPSB, CMPSD, CMPSW, SCAS, SCASB, SCASD, and SCASW instructions. Table 1-8 shows the valid REPNE and REPNZ prefix opcodes.

Table 1-8. REPNE and REPNZ Opcodes

Mnemonic	Opcode
REPNe CMPS <i>mem8, mem8</i> REPNe CMPSB	F2 A6
REPNe CMPS <i>mem16/32/64, mem16/32/64</i> REPNe CMPSW REPNe CMPSD REPNe CMPSQ	F2 A7
REPNe SCAS <i>mem8</i> REPNe SCASB	F2 AE
REPNe SCAS <i>mem16/32/64</i> REPNe SCASW REPNe SCASD REPNe SCASQ	F2 AF

Instructions that Cannot Use Repeat Prefixes. The repeat prefixes should only be used in string instructions listed above. When used in 128-bit or 64-bit media instructions, the F2h and F3h prefixes act in a special way to modify the opcode rather than cause a repeat operation. The result of using a 66h operand-size prefix along with a F2h or F3h prefix in 128-bit or 64-bit media instructions is unpredictable.

Optimization of Repeats. Depending on the hardware implementation, the repeat prefixes can have a setup overhead. If the repeated count is variable, the overhead can sometimes be avoided by substituting a simple loop to move or store the data. Repeated string instructions can be expanded into equivalent sequences of inline loads and stores, or a sequence of stores can be used to emulate a REP STOS.

For repeated string moves, performance can be maximized by moving the largest possible operand size. For example, use REP MOVSD rather than REP MOVSW and REP MOVSW rather than REP MOVSB. Use REP STOSD rather than REP STOSW and REP STOSW rather than REP MOVSB.

Depending on the hardware implementation, string moves with the direction flag (DF) cleared to 0 (up) may be faster than string moves with DF set to 1 (down). DF = 1 is only needed for certain cases of overlapping REP MOVSB, such as when source and destination overlap.

1.2.7 REX Prefixes

REX prefixes are a group of instruction-prefix bytes that can be used only in 64-bit mode. They enable access to the x86-64 register extensions. Figure 1-1 on page 1 and Figure 1-2 on page 2 show how a REX prefix fits within the byte order of instructions. REX prefixes enable the following features in 64-bit mode:

- Use of extended GPR (Figure 1-3 on page 29) or XMM registers (Figure 1-6 on page 32).
- Use of 64-bit operand size when accessing GPRs.
- Use of RIP-relative addressing, as described in Section 2.2.5, “RIP-Relative Addressing”, on page 21 of Volume 1.
- Use of extended control and debug registers, as described in Section 3.1.5, “64-Bit-Mode Extended Control Registers”, on page 63 of Volume 2 and “64-Bit-Mode Extended Debug Registers” on page 403 of Volume 2.

Table 1-9 shows the REX prefixes. The value of a REX prefix is in the range 40h to 4Fh, depending on the particular combination of x86-64 register extensions desired.

Table 1-9. REX Instruction Prefixes

Prefix Type	Mnemonic	Prefix Code (Hex)	Description
Register Extensions	REX.W	40 ¹ through 4F ¹	Access an x86-64 register extension.
	REX.R		
	REX.X		
	REX.B		
Note: 1. See Table 1-11 for encoding of REX prefixes.			

A REX prefix is normally required with an instruction that accesses a 64-bit GPR or one of the extended GPR or XMM registers. Only a few instructions have an operand size that defaults to (or is fixed at) 64 bits in 64-bit mode, and thus do not need a REX prefix. These exceptions to the normal rule are listed in Table 1-9.

Table 1-10. Instructions Not Requiring REX Prefix in 64-Bit Mode

CALL near	LOOPcc
CMPXCHG8B	MOV to/from Control Register
Jcc	MOV to/from Debug Register
JCXZ, JECXZ, JRCXZ	RET
JMP near	

An instruction can have only one REX prefix, although the prefix can express several extension features. If a REX prefix is used, it must immediately precede the first opcode byte in the instruction format. Any other placement of a REX prefix, or any use of a REX prefix in an instruction that does not access an extended register, is ignored. The legacy instruction-size limit of 15 bytes still applies to instructions that contain a REX prefix.

REX prefixes are a set of sixteen values that span one row of the main opcode map and occupy entries 40h to 4Fh. Table 1-11 and Figure 1-3 show the prefix fields and their uses.

Table 1-11. REX Prefix-Byte Fields

Mnemonic	Bit Position	Definition
—	7:4	0100
REX.W	3	0 = Default operand size 1 = 64-bit operand size
REX.R	2	1-bit (high) extension of the ModRM <i>reg</i> field ¹ , thus permitting access to 16 registers.
REX.X	1	1-bit (high) extension of the SIB <i>index</i> field ¹ , thus permitting access to 16 registers.
REX.B	0	1-bit (high) extension of the ModRM <i>r/m</i> field ¹ , SIB <i>base</i> field ¹ , or opcode <i>reg</i> field, thus permitting access to 16 registers.
Note: 1. See Section 1.4, “ModRM and SIB Bytes”, on page 20, for a description of the ModRM and SIB bytes.		

REX.W: Operand Width. Setting the REX.W bit to 1 specifies a 64-bit operand size. Like the existing 66h operand size prefix, the REX 64-bit operand size override has no effect on byte operations. For non-byte operations, the REX operand-size override takes precedence over the 66h prefix. If a 66h prefix is used together with a REX prefix that has the REX.W bit set to 1, the 66h prefix is ignored. However, if a 66h prefix is used together with a REX prefix that has the REX.W bit cleared to 0, the 66h prefix is not ignored and the operand size becomes 16 bits.

REX.R: Register. The REX.R bit adds a 1-bit (high) extension to the ModRM *reg* field (page 20) when that field encodes a GPR or XMM register. REX.R does not modify ModRM *reg* when that field specifies other registers or opcodes. REX.R is ignored in such cases.

REX.X: Index. The REX.X bit adds a 1-bit (high) extension to the SIB *index* field (page 20).

REX.B: Base. The REX.B bit either adds a 1-bit (high) extension to the base in the ModRM *r/m* field or SIB *base* field, or it adds a 1-bit (high) extension to the opcode *reg* field used for accessing

GPRs, control registers, or debug registers. (See Table 2-2 on page 45 for more about the REX.B bit.)

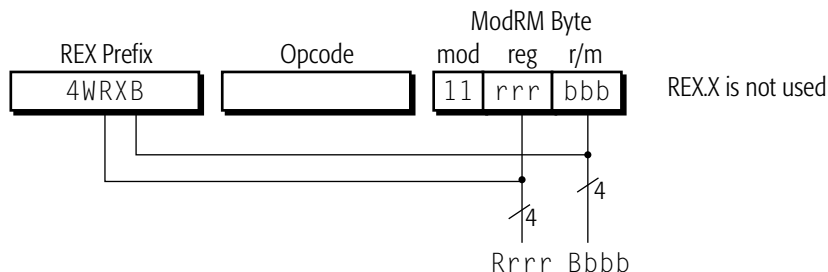
Encoding Examples. Figure 1-3 shows four examples of how the R, X, and B bits of REX prefixes are concatenated with fields from the ModRM byte, SIB byte, and opcode to specify register and memory addressing. The R, X, and B bits are described in Table 1-11 on page 16.

Byte-Register Addressing. In the legacy architecture, the byte registers (AH, AL, BH, BL, CH, CL, DH, and DL, shown in Figure 1-2 on page 28) are encoded in the ModRM *reg* or *r/m* field or in the opcode *reg* field as registers 0 through 7. The REX prefix provides an additional byte-register addressing capability that makes the least-significant byte of any GPR available for byte operations (Figure 1-3 on page 29). This provides a uniform set of byte, word, doubleword, and quadword registers better suited for register allocation by compilers.

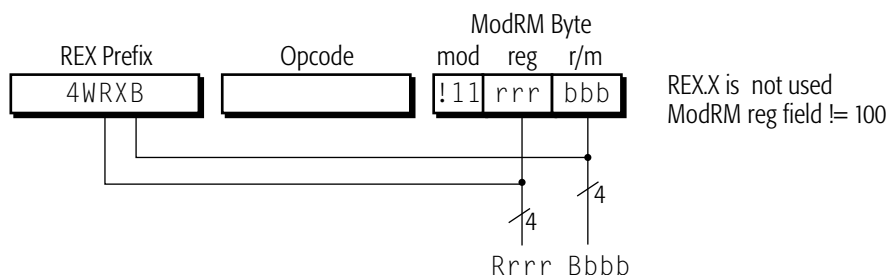
Special Encodings for Registers. Readers who need to know the details of instruction encodings should be aware that certain combinations of the ModRM and SIB fields have special meaning for register encodings. For some of these combinations, the instruction fields expanded by the REX prefix are not decoded (treated as don't cares), thereby creating aliases of these encodings in the extended registers. Table 1-12 on page 19 describes how each of these cases behaves.

Implications for INC and DEC Instructions. The REX prefix values are taken from the 16 single-byte INC and DEC instructions, one for each of the eight GPRs. Therefore, these single-byte opcodes for INC and DEC are not available in 64-bit mode, although they are available in legacy and compatibility modes. The functionality of these INC and DEC instructions is still available in 64-bit mode, however, using the ModRM forms of those instructions (opcodes FF/0 and FF/1).

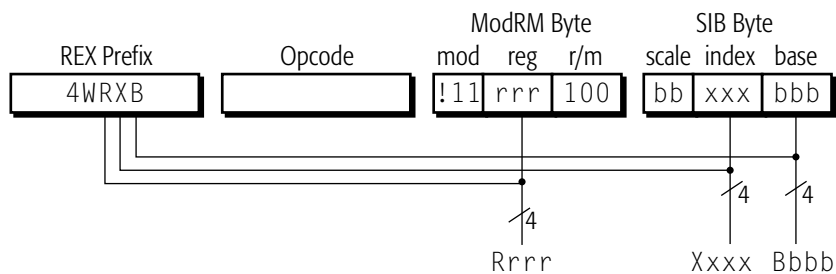
Case 1: Register-Register Addressing (No Memory Operand)



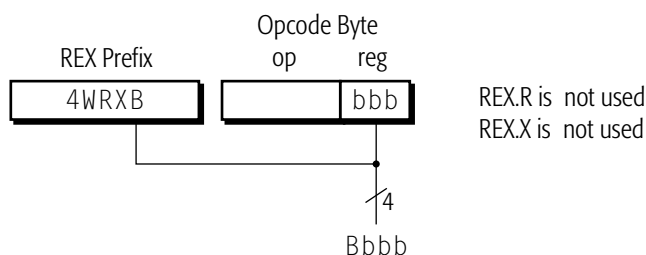
Case 2: Memory Addressing Without an SIB Byte



Case 3: Memory Addressing With an SIB Byte



Case 4: Register Operand Coded in Opcode Byte



513-302.eps

Figure 1-3. Encoding Examples of REX-Prefix R, X, and B Bits

Table 1-12. Special REX Encodings for Registers

ModRM and SIB Encodings ²	Meaning in Legacy and Compatibility Modes	Implications in Legacy and Compatibility Modes	Additional REX Implications
ModRM Byte: • $\text{mod} \neq 11$ • $r/m^1 = \text{x100}$ (ESP)	SIB byte is present.	SIB byte required for ESP-based addressing.	REX prefix adds a fourth bit (B), which is not decoded (don't care). Therefore, SIB byte also required for R12-based addressing.
ModRM Byte: • $\text{mod} = 00$ • $r/m^1 = \text{x101}$ (EBP)	Base register is not used.	Using EBP without displacement must be done via $\text{mod} = 01$ with a displacement of 0 (with or without an index register).	REX prefix adds a fourth bit (B), which is not decoded (don't care). Therefore, using RBP or R13 without displacement must be done via $\text{mod} = 01$ with a displacement of 0 (with or without an index register).
SIB Byte: • $\text{index}^1 = 0100$ (ESP)	Index register is not used.	ESP cannot be used as an index register.	REX prefix adds a fourth bit (X), which is decoded. Therefore, there are no additional implications. The expanded index field is used to distinguish RSP from R12, allowing R12 to be used as an index.
Note: 1. The REX-prefix bit is shown in the fourth (most-significant) bit position of the encodings for the ModRM r/m , SIB index, and SIB base fields. The lower-case "x" for ModRM r/m (rather than the upper-case "B" shown in Figure 1-3 on page 18) indicates that the REX-prefix bit is not decoded (don't care). 2. See Section 1.4, "ModRM and SIB Bytes", on page 20, for a description of the ModRM and SIB bytes.			

1.3 Opcode

An instruction's opcode is unique for each instruction, although assemblers can support multiple mnemonics for a single instruction opcode. The opcode specifies the operation it performs and, in certain cases, the kinds of operands it uses. An opcode consists of 1 or 2 bytes, but certain 128-bit media instructions also use a prefix byte in a special way to modify the opcode. The 3-bit *reg* field of the ModRM byte (Section 1.4, "ModRM and SIB Bytes", on page 20) is also used in certain

instructions either for 3 additional opcode bits or for a register specification.

128-Bit and 64-Bit Media Instruction Opcodes. Many 128-bit and 64-bit media instructions include a 66h, F2h, or F3h prefix byte in a special way to modify the opcode. These same byte values can be used in certain general-purpose and x87 instructions to modify operand size (66h) or repeat the function (F2h, F3h). In 128-bit and 64-bit media instructions, however, such prefix bytes modify the opcode. If a 128-bit and 64-bit media instruction uses one of these three prefixes, and also includes any other prefix in the 66h, F2h, and F3h group, the result is unpredictable.

All opcodes for 64-bit media instructions begin with an 0Fh byte. In the case of 64-bit floating-point (3DNow!) instructions, the 0Fh byte is followed by a second 0Fh opcode byte. A third opcode byte occupies the same position at the end of a 3DNow! instruction as would an immediate byte. The value of the immediate byte is shown as the third opcode byte-value in the syntax for each instruction in Section 2.8, “64-Bit Media Instruction Reference”, on page 787. The format is:

0Fh 0Fh ModRM [SIB] [displacement] 3DNow!_third_opcode_byte

For details on opcode encoding, see Section 3, “Opcode and Operand Encodings”, on page 1123.

1.4 ModRM and SIB Bytes

The ModRM byte is used in certain instruction encodings to:

- Define a register reference.
- Define a memory reference.
- Provide additional opcode bits with which to define the instruction’s function.

ModRM bytes have three fields—*mod*, *reg*, and *r/m*. The *reg* field provides additional opcode bits with which to define the instruction’s function. The *mod* and *r/m* fields are used together with each other and, in 64-bit mode, with the REX.R and REX.B bits of the REX prefix (page 14), to specify the location of the instruction’s operands and certain of the possible addressing modes (specifically, the non-complex modes).

Figure 1-4 shows the format of a ModRM byte.

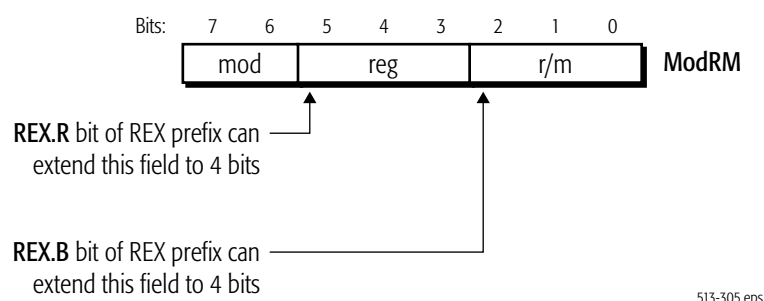


Figure 1-4. ModRM-Byte Format

In some instructions, the ModRM byte is followed by a SIB byte, which defines memory addressing for the complex-addressing modes described in Section 2.2.3, “Effective Addresses”, on page 17 of Volume 1. The SIB byte has three fields—*scale*, *index*, and *base*—that define the scale factor, index-register number, and base-register number for 32-bit and 64-bit complex addressing modes. In 64-bit mode, the REX.B and REX.X bits extend the encoding of the SIB byte’s *base* and *index* fields.

Figure 1-5 shows the format of a SIB byte.

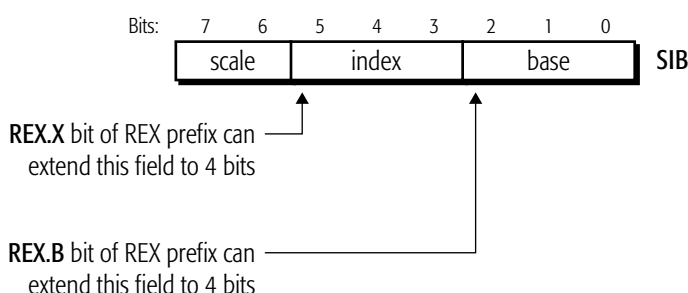


Figure 1-5. SIB-Byte Format

The encodings of ModRM and SIB bytes not only define memory-addressing modes, but they also specify operand registers. The encodings do this by using 3-bit fields in the ModRM and SIB bytes, depending on the format:

- *ModRM*: the *reg* and *r/m* fields of the ModRM byte. (Case 1 in Figure 1-3 on page 18 shows an example of this).

- *ModRM With SIB*: the *reg* field of the ModRM byte and the *base* and *index* fields of the SIB byte. (Case 3 in Figure 1-3 on page 18 shows an example of this).
- *Instructions Without ModRM*: the *reg* field of the opcode. (Case 4 in Figure 1-3 on page 18 shows an example of this).

In 64-bit mode, the bits needed to extend each field for accessing the additional registers are provided by the REX prefixes, as shown in Figure 1-4 and Figure 1-5.

For details on opcode encoding, see Section 3, “Opcode and Operand Encodings”, on page 1123.

1.5 Displacement Bytes

A *displacement* (also called *offset*) is a signed value that is added to the base of a code segment (absolute addressing) or to an instruction pointer (relative addressing), depending on the addressing mode. In general, the size of a displacement is 1, 2, or 4 bytes. If an addressing mode requires a displacement, the bytes (1, 2, or 4) for the displacement follow the opcode, ModRM, or SIB byte (whichever comes last) in the instruction encoding.

In 64-bit mode, the same ModRM and SIB encodings are used for displacement sizes as are used in legacy and compatibility modes. However, the displacement is sign-extended to 64 bits during effective-address calculations. Also, in 64-bit mode, support is provided for some 64-bit displacement and immediate forms of the MOV instruction. See “Immediate Operand Size” on page 47 of Volume 1 and Section 2.2.6, “Direct-Memory-Offset Moves”, on page 21 of Volume 1 for more information on this.

1.6 Immediate Bytes

An *immediate* is a value—typically an operand value—encoded directly into the instruction. Depending on the opcode and the operating mode, the size of an immediate operand can be 1, 2, or 4 bytes. Immediate operands in 64-bit mode are limited to these same sizes.

If an instruction takes an immediate operand, the bytes (1, 2, or 4) for the immediate follow the opcode, ModRM, SIB, or

displacement bytes (whichever come last) in the instruction encoding. Some 128-bit media instructions use the immediate byte as a condition code.

1.7 RIP-Relative Addressing

In 64-bit mode, addressing relative to the contents of 64-bit instruction pointer (program counter)—called RIP-relative addressing or PC-relative addressing—is implemented for certain instructions. In such cases, the effective address is formed by adding the displacement to the 64-bit RIP of the next instruction.

In the legacy x86 architecture, addressing relative to the instruction pointer is available only in control-transfer instructions. In the 64-bit mode, any instruction that uses ModRM addressing can use RIP-relative addressing. The feature is particularly useful for addressing data in position-independent code and for code that addresses global data.

Without RIP-relative addressing, ModRM instructions address memory relative to zero. With RIP-relative addressing, ModRM instructions can address memory relative to the 64-bit RIP using a signed 32-bit displacement. This provides an offset range of $\pm 2\text{GB}$ from the RIP.

Programs usually have many references to data, especially global data, that are not register-based. To load such a program, the loader typically selects a location for the program in memory and then adjusts the program's references to global data based on the load location. RIP-relative addressing of data makes this adjustment unnecessary.

1.7.1 Encoding

Table 1-13 shows the ModRM and SIB encodings for RIP-relative addressing. Redundant forms of 32-bit displacement-only addressing exist in the current ModRM and SIB encodings. There is one ModRM encoding and several SIB encodings. RIP-relative addressing is encoded using one of the redundant forms. In 64-bit mode, the ModRM *Disp32* (32-bit displacement) encoding is re-defined to be $RIP + Disp32$ rather than displacement-only.

Table 1-13. Encoding for RIP-Relative Addressing

ModRM and SIB Encodings	Meaning in Legacy and Compatibility Modes	Meaning in 64-bit Mode	Additional 64-bit Implications
ModRM Byte: <ul style="list-style-type: none"> mod = 00 r/m = 101 (none) 	Disp32	RIP + Disp32	Zero-based (normal) displacement addressing must use SIB form (see next row).
SIB Byte: <ul style="list-style-type: none"> base = 101 (none) index = 100 (none) scale = 0, 1, 2, 4 	If mod = 00, Disp32	Same as Legacy	None

1.7.2 REX Prefix and RIP-Relative Addressing

ModRM encoding for RIP-relative addressing does not depend on a REX prefix. In particular, the *r/m* encoding of 101, used to select RIP-relative addressing, is not affected by the REX prefix. For example, selecting R13 (REX.B=1, *r/m*=101) with mod=00 still results in RIP-relative addressing.

The 4-bit *r/m* field of ModRM is not fully decoded. Therefore, in order to address R13 with no displacement, software must encode it as R13+0 using a 1-byte displacement of zero.

1.7.3 Address-Size Prefix and RIP-Relative Addressing

RIP-relative addressing is enabled by 64-bit mode, not by a 64-bit address-size. Conversely, use of the address-size prefix (Section 1.2.3 on page 6) does not disable RIP-relative addressing. The effect of the address-size prefix is to truncate and zero-extend the computed effective address to 32 bits, like any other addressing mode.

2 Instruction Details

2.1 Instruction Subsets

This section describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by all instructions in the x86-64 architecture. The descriptions are divided into five instruction subsets:

- *Section 2.6, “General-Purpose Instruction Reference”, on page 57*—General-purpose instructions are used in basic software execution. Most of them load, store, or operate on data in the general-purpose registers (GPRs), in memory, or in both. The remaining instructions are used to alter sequential program flow by branching to other locations within the program or to entirely different programs.
- *Section 2.7, “128-Bit Media Instruction Reference”, on page 291*—128-bit media instructions load, store, or operate on data located in the 128-bit XMM registers. The instructions define both vector and scalar operations on floating-point and integer data types. They include the SSE and SSE2 instructions that operate on the XMM registers. Some of the instructions convert source operands in XMM registers to destination operands in GPR, MMX™, or x87 registers or otherwise affect XMM state.
- *Section 2.8, “64-Bit Media Instruction Reference”, on page 681*—64-bit media instructions load, store, or operate on data located in the 64-bit MMX registers. The instructions define both vector and scalar operations on integer and floating-point data types. They include the legacy MMX instructions, the 3DNow!™ instructions, and the AMD extensions to the MMX and 3DNow! instruction sets. Some of the instructions convert source operands in MMX registers to destination operands in GPR, XMM, or x87 registers or otherwise affect MMX state.
- *Section 2.9, “x87 Floating-Point Instruction Reference”, on page 909*—The x87 instructions are used in legacy floating-point applications. Most of these instructions load, store, or operate on data located in the x87 ST(0)–ST(7) stack registers (the FPR0–FPR7 physical registers). The remaining instructions within this category are used to manage the x87 floating-point environment.

- Section 2.10, “System Instruction Reference”, on page 1047—The system instructions are used to establish the processor’s operating mode, access processor resources, handle program and system errors, and manage memory.

Instructions that apply to two or more of the above subsets have their descriptions repeated identically in all such subsets. The description of each instruction covers its behavior in all operating modes, including legacy mode (real, virtual-8086, and protected modes) and long mode (compatibility and 64-bit modes). Details of certain kinds of complex behavior—such as control-flow changes in CALL, INT, or FXSAVE instructions—have cross-references in the instruction-detail pages to detailed descriptions elsewhere in the other two volumes of this manual.

Two instructions—CMPSD and MOVSD—use the same mnemonic for different instructions. Assemblers can distinguish them on the basis of the number and type of operands with which they are used.

2.2 Reference-Page Format

Figure 2-1 shows the format of an instruction-detail page. The instruction mnemonic is shown in bold at the top-left, along with its English name. In this example, **POPFD** is the mnemonic and *POP to EFLAGS Doubleword* is the name. Next, there is a general description of the instruction’s operation. Many descriptions have cross-references to more detail in other parts of the manual.

Beneath the general description, the mnemonic is shown again, together with the related opcode(s) and a description summary. Related instructions are listed below this, followed by a table showing the flags that the instruction can affect. Finally, each instruction has a summary of the possible exceptions that can occur when executing the instruction. The columns labeled “Real” and “Virtual-8086” apply only to execution in legacy mode. The column labeled “Protected” applies both to legacy mode and long mode, because long mode is a superset of legacy protected mode.

128-bit and 64-bit media instructions also have diagrams illustrating the operation. A few instructions have examples or pseudocode describing the action.

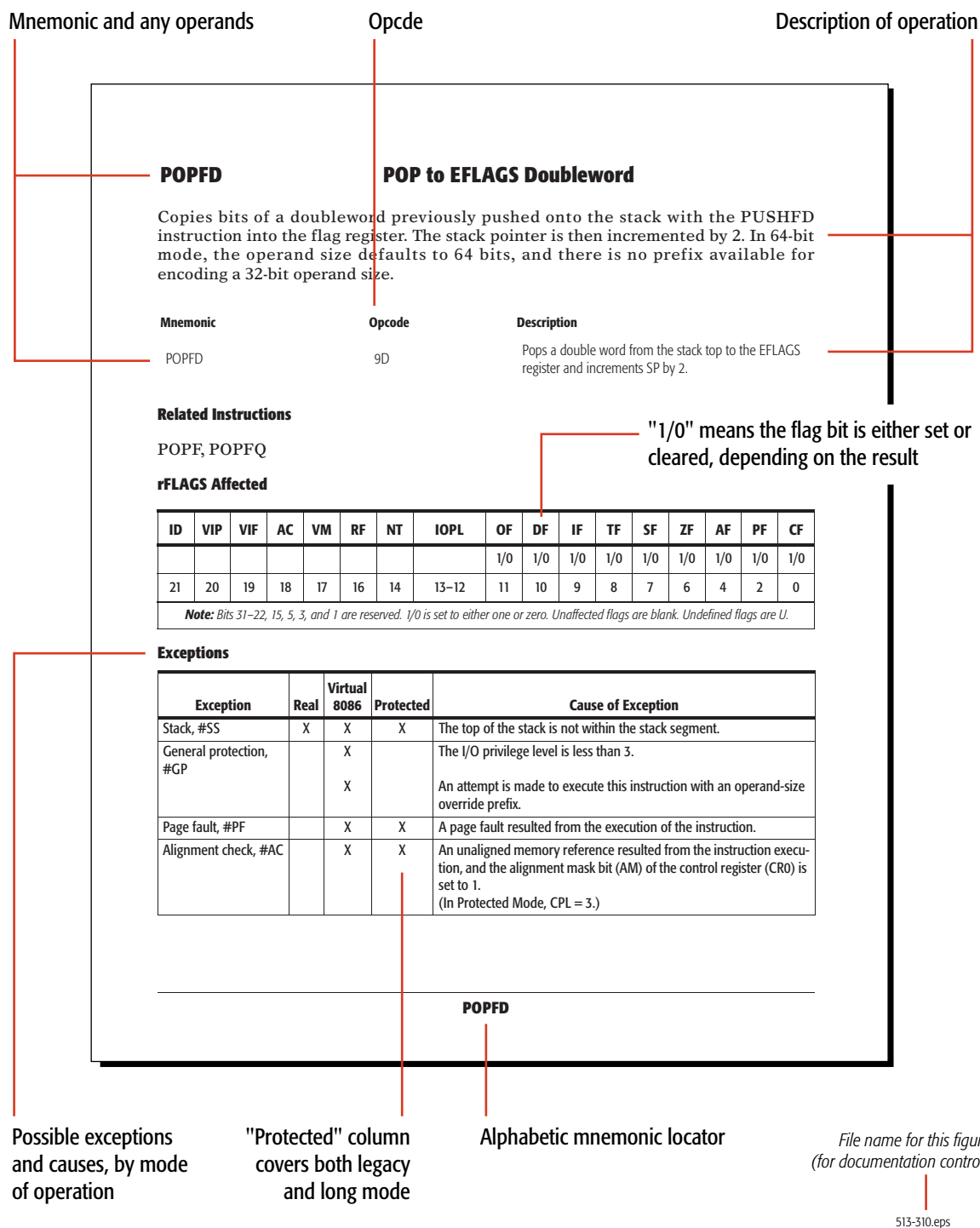


Figure 2-1. Format of Instruction-Detail Pages

2.3 Summary of Registers and Data Types

This section summarizes the registers available to software using the five instruction subsets described in Section 2.1, “Instruction Subsets”, on page 25. For details on the organization and use of these registers, see their respective chapters in Volumes 1 and 2.

2.3.1 General-Purpose Instructions

Registers. The size and number of general-purpose register (GPRs) depends on the operating mode, as do the size of the flags and instruction-pointer registers. Figure 2-2 shows the registers available in legacy and compatibility modes.

register encoding	high 8-bit		low 8-bit	16-bit	32-bit
0		AH (4)	AL	AX	EAX
3		BH (5)	BL	BX	EBX
1		CH (7)	CL	CX	ECX
2		DH (6)	DL	DX	EDX
6			SI	SI	ESI
7			DI	DI	EDI
5			BP	BP	EBP
4			SP	SP	ESP
31		16 15		0	

		FLAGS	FLAGS	EFLAGS
		IP	IP	EIP
31				0

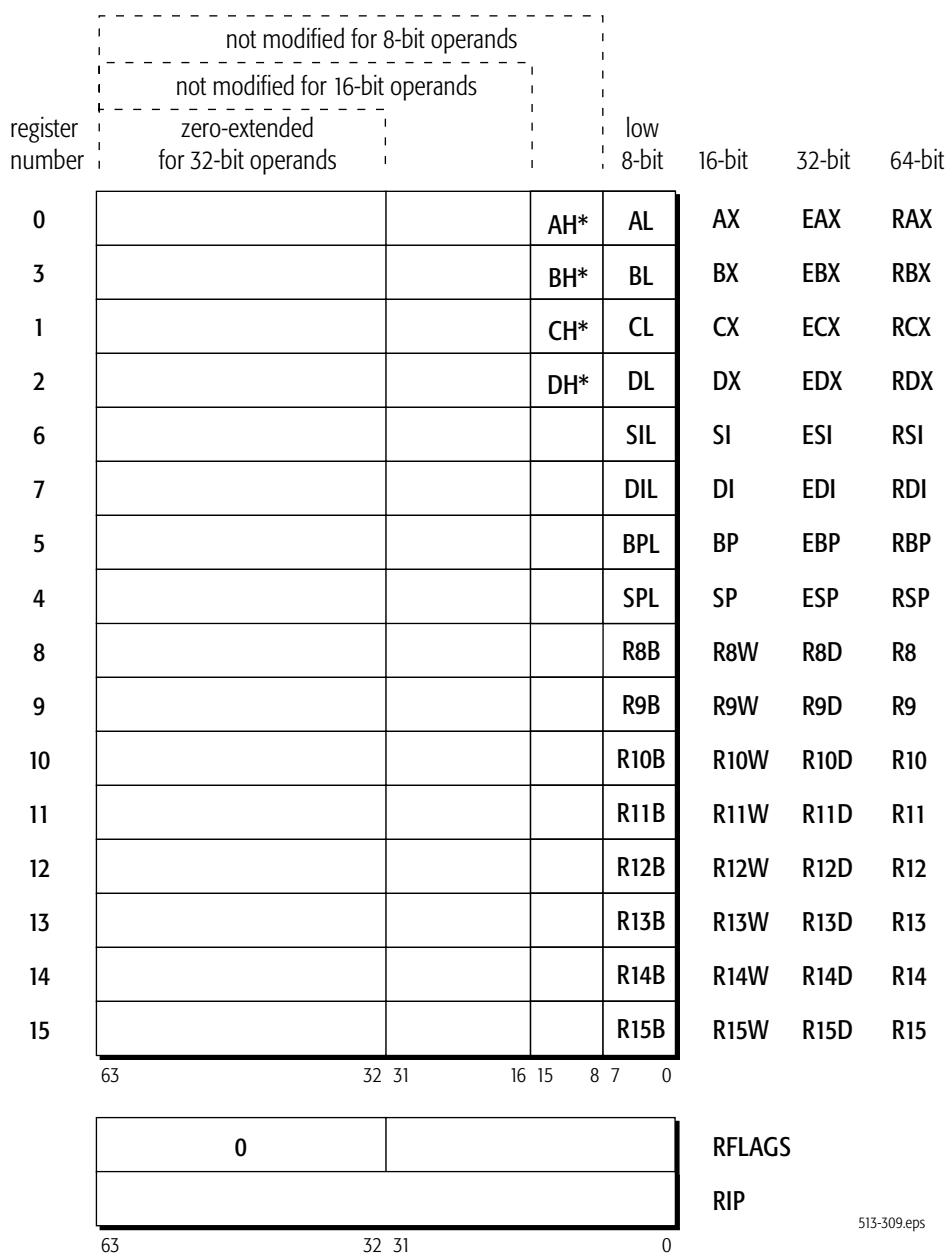
Registers are ordered according to mnemonic.
This ordering does not correspond exactly to
register encoding within instructions.

513-311.eps

Figure 2-2. General Registers in Legacy and Compatibility Modes

Figure 2-3 shows the registers accessible in 64-bit mode. Compared with legacy mode, registers become 64 bits wide, eight new data registers (R8–R15) are added, the low byte of all 16 GPRs is available for byte operations, and the four high-byte registers of legacy mode (AH, BH, CH, and DH) are not

available. The high 32 bits of doubleword operands are zero-extended to 64 bits, but the high bits of word and byte operands are not modified by operations in 64-bit mode. The RFLAGS register is 64 bits wide, but the high 32 bits are reserved. They can be written with anything but they read as zeros (RAZ).



Registers are ordered according to mnemonic. This ordering does not correspond exactly to register encoding within instructions.

* Not addressable when a REX prefix is used.

Figure 2-3. General Registers in 64-Bit Mode

For most instructions running in 64-bit mode, access to the extended GPRs requires a REX instruction prefix (page 14).

Figure 2-4 shows the segment registers which, like the instruction pointer, are used by all instructions. In legacy and compatibility modes, all segments are accessible. In 64-bit mode, which uses the flat (non-segmented) memory model, only the CS, FS, and GS segments are recognized, whereas the contents of the DS, ES, and SS segment registers are ignored (the base for each of these segments is assumed to be zero, and neither their segment limit nor attributes are checked). For details, see Section 4, “Segmented Virtual Memory”, on page 89 of Volume 2.

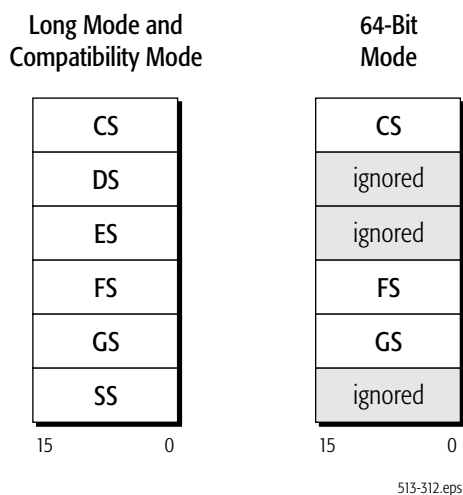


Figure 2-4. Segment Registers

Data Types. Figure 2-2 shows the general-purpose data types. They are all scalar, integer data types. The 64-bit (quadword) data types are only available in 64-bit mode, and for most instructions they require a REX instruction prefix.

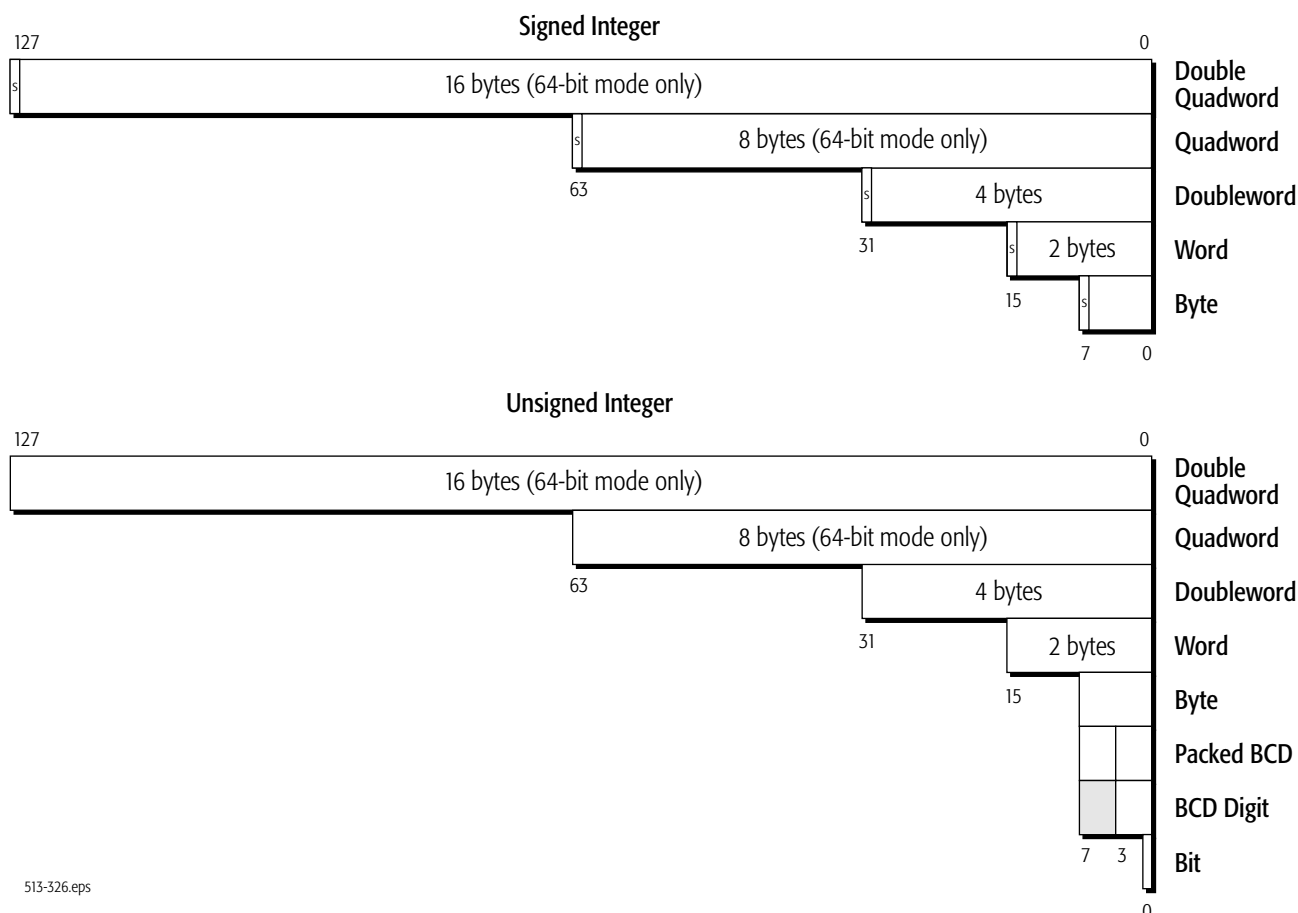


Figure 2-5. General-Purpose Data Types

2.3.2 128-Bit Media Instructions

Registers. 128-bit media instructions use the 128-bit XMM registers. The number of available XMM data registers depends on the operating mode, as shown in Figure 2-6. In legacy and compatibility modes, the eight legacy XMM data registers (XMM0–XMM7) are available. In 64-bit mode, eight additional XMM data registers (XMM8–XMM15) are available when a REX instruction prefix is used.

The MXCSR register contains floating-point and other control and status flags used by the 128-bit media instructions. Some 128-bit media instructions also use the GPR (Figure 2-2 and Figure 2-3) and the MMX registers (Figure 2-8) or set or clear flags in the rFLAGS register (see Figure 2-2 and Figure 2-3).

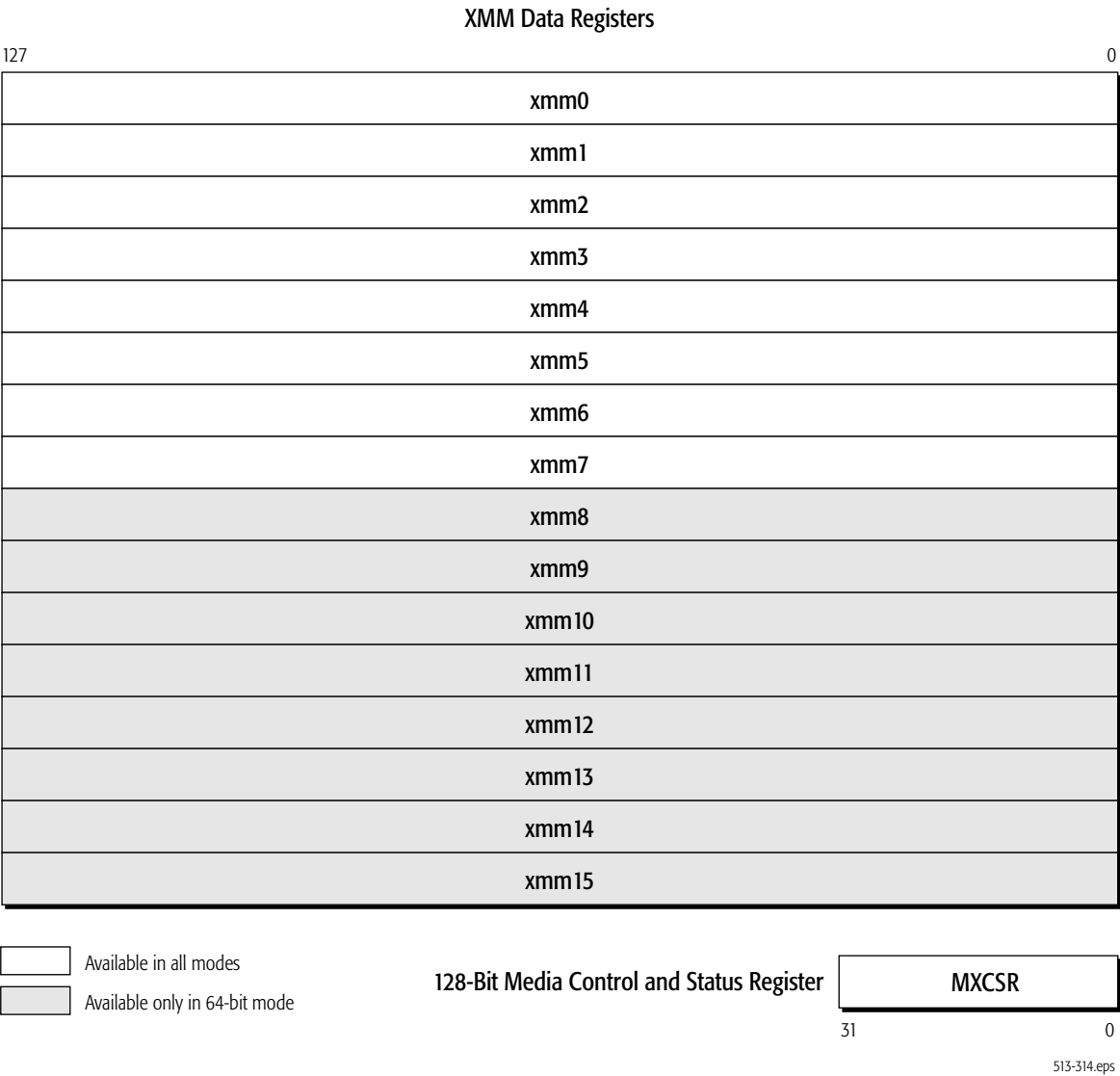
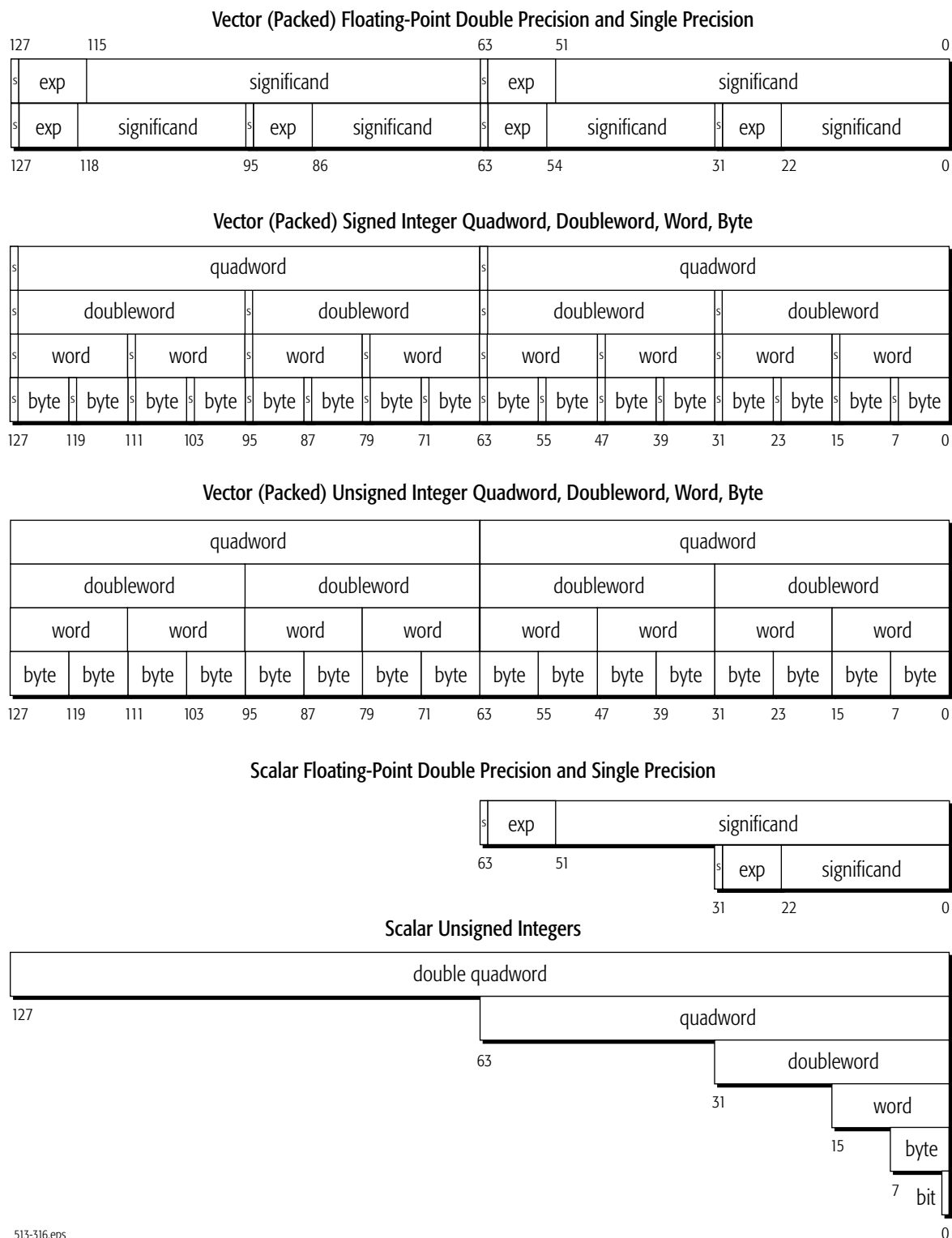


Figure 2-6. 128-Bit Media Registers

Data Types. Figure 2-7 shows the 128-bit media data types. They include floating-point and integer vectors and floating-point scalars. The floating-point data types include IEEE-754 single precision and double precision.

**Figure 2-7. 128-Bit Media Data Types**

2.3.3 64-Bit Media Instructions

Registers. 64-bit media instructions use the eight 64-bit MMX registers, as shown in Figure 2-8. These registers are mapped onto the x87 floating-point registers, and 64-bit media instructions write the x87 tag word in a way that prevents an x87 instruction from using MMX data.

Some 64-bit media instructions also use the GPR (Figure 2-2 and Figure 2-3) and the XMM registers (Figure 2-6).

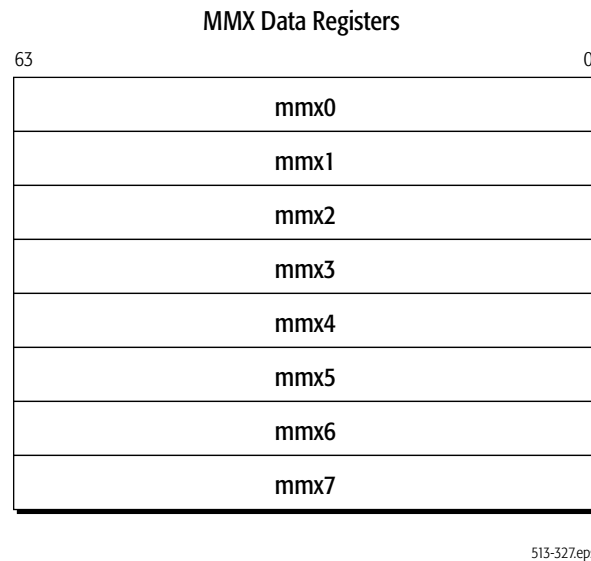


Figure 2-8. 64-Bit Media Registers

Data Types. Figure 2-9 shows the 64-bit media data types. They include floating-point and integer vectors and integer scalars. The floating-point data type, used by 3DNow! instructions, consists of a packed vector or two IEEE-754 32-bit single-precision data types. Unlike other kinds of floating-point instructions, however, the 3DNow! instructions do not generate floating-point exceptions. For this reason, there is no register for reporting or controlling the status of exceptions in the 64-bit media instruction subset.

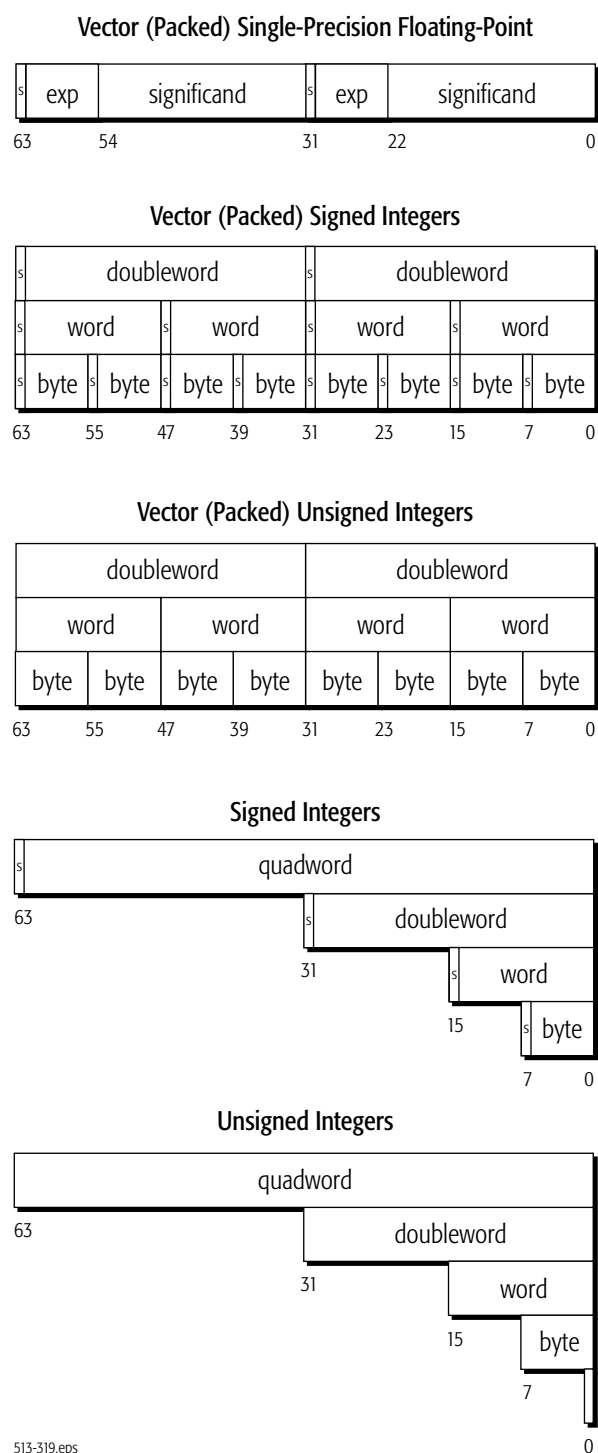


Figure 2-9. 64-Bit Media Data Types

2.3.4 x87 Floating-Point Instructions

Registers. x87 floating-point instructions use the x87 registers shown in Figure 2-10. There are eight 80-bit data registers, three 16-bit registers that hold the x87 control word, status word, and tag word, and three registers (last instruction pointer, last opcode, last data pointer) that hold information about the last x87 operation.

The physical data registers are named FPR0–FPR7, although x87 software references these registers as a stack of registers, named ST(0)–ST(7). x87 instructions store operands only in their own 80-bit floating-point registers or in memory. They do not access the GPR, XMM, or MMX registers.

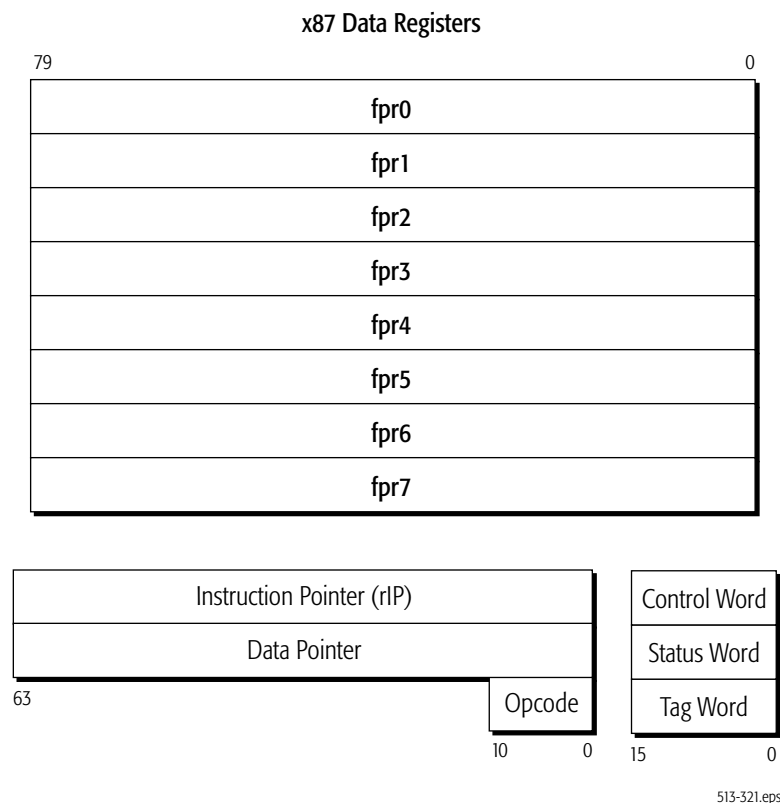
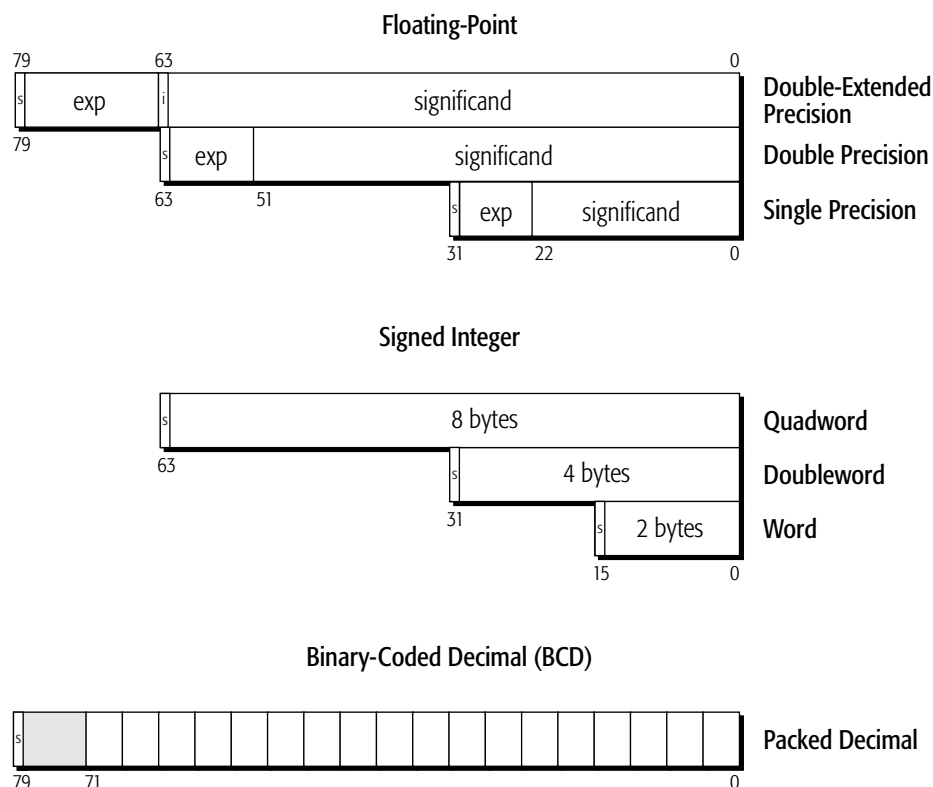


Figure 2-10. x87 Registers

Data Types. Figure 2-11 shows the x87 data types. They include three floating-point formats (80-bit double-extended precision, 64-bit double precision, and 32-bit single precision)—three signed-integer formats (quadword, doubleword, and word), and an 80-bit packed binary-coded decimal (BCD) format.



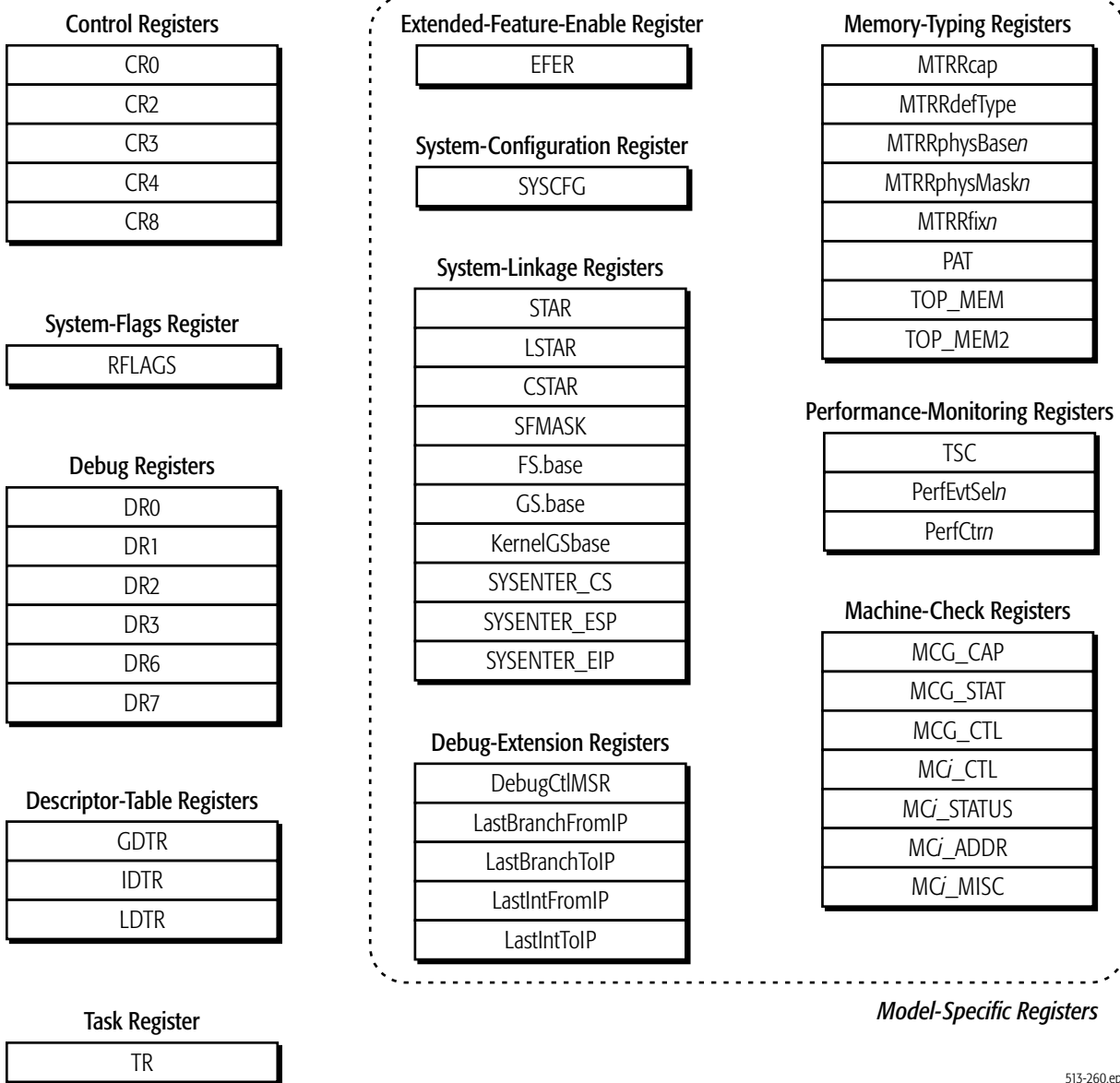
513-317eps

Figure 2-11. x87 Data Types

2.3.5 System Instructions

Registers. The system instructions use several specialized registers shown in Figure 2-12. System software uses these registers to, among other things, manage the processor's operating environment, define system resource characteristics, and to monitor software execution. With the exception of the RFLAGS register, system registers can be read and written only from privileged software.

All system registers are 64 bits wide, except for the descriptor-table registers and the task register, which include 64-bit base-address fields and other fields.



513-260.eps

Figure 2-12. System Registers

Data Structures. Figure 2-13 shows the system data structures. These are created and maintained by system software for use in protected mode. A processor running in protected mode uses these data structures to manage memory and protection, and to store program-state information when an interrupt or task switch occurs.

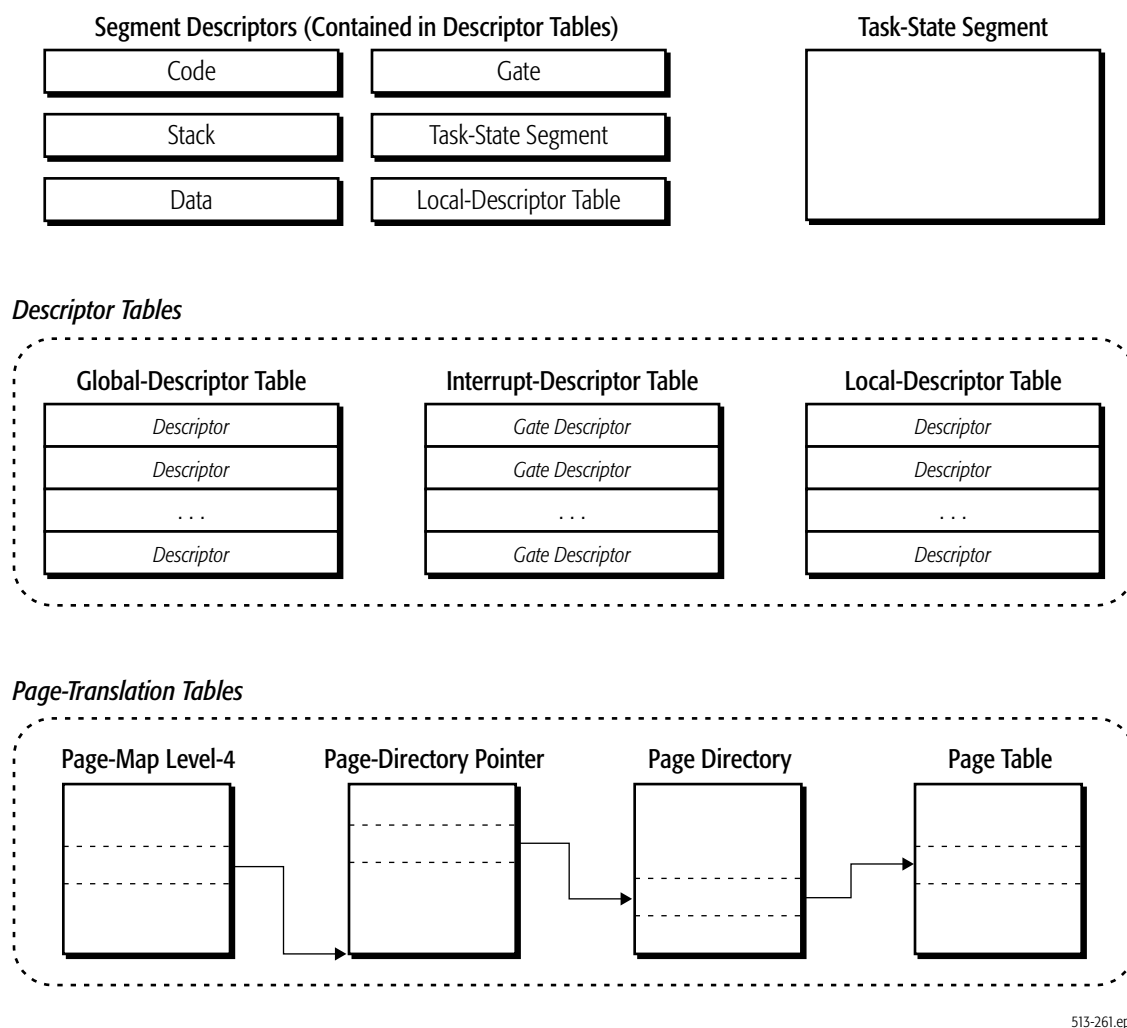


Figure 2-13. System Data Structures

2.4 Summary of Exceptions

Table 2-1 lists all exceptions that are possible in legacy protected mode and in long mode. The table shows the interrupt-vector numbers, names, mnemonics, source, and possible causes. Exceptions that apply to specific instructions are documented with each instruction in the instruction-detail pages that follow.

Table 2-1. Interrupt-Vector Source and Cause

Vector	Interrupt (Exception)	Mnemonic	Source	Cause
0	Divide-By-Zero-Error	#DE	Software	DIV, IDIV instructions
1	Debug	#DB	Internal	Instruction accesses and data accesses
2	Non-Maskable-Interrupt	—	External	External NMI signal
3	Breakpoint	#BP	Software	INT3 instruction
4	Overflow	#OF	Software	INTO instruction
5	Bound-Range	#BR	Software	BOUND instruction
6	Invalid-Opcode	#UD	Internal	Invalid instructions
7	Device-Not-Available	#NM	Internal	x87 instructions
8	Double-Fault	#DF	Internal	Interrupt during an interrupt
9	Coprocessor-Segment-Overrun	—	External	Unsupported (reserved)
10	Invalid-TSS	#TS	Internal	Task-state segment access and task switch
11	Segment-Not-Present	#NP	Internal	Segment access through a descriptor
12	Stack	#SS	Internal	SS register loads and stack references
13	General-Protection	#GP	Internal	Memory accesses and protection checks
14	Page-Fault	#PF	Internal	Memory accesses when paging enabled
15	Reserved	—		
16	Floating-Point Exception-Pending	#MF	Software	x87 floating-point and 64-bit media floating-point instructions
17	Alignment-Check	#AC	Internal	Memory accesses
18	Machine-Check	#MC	Internal External	Model specific
19	SIMD Floating-Point	#XF	Internal	128-bit media floating-point instructions
20–31	Reserved (Internal and External)	—		
32–255	External Interrupts (Maskable)	—	External	External interrupt signalling
0–255	Software Interrupts	—	Software	INT n instruction

2.5 Notation

2.5.1 Mnemonic Syntax

Each instruction has a mnemonic syntax that includes the mnemonic and any operands that the instruction can take. Figure 2-14 shows an example of mnemonic syntax in which the instruction takes two operands. In most instructions that take two operands, the first (left-most) operand is both a source operand (the first source operand) and the destination operand. The second (right-most) operand serves only as a source, not a destination.

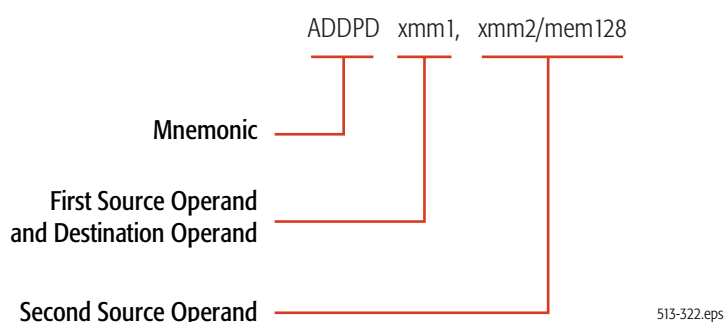


Figure 2-14. Mnemonic Syntax for Typical Two-Operand Instruction

The following notation is used to denote the size and type and source and destination operands:

- *cReg*—Control register.
- *dReg*—Debug register.
- *imm8*—Byte (8-bit) immediate.
- *imm16*—Word (16-bit) immediate.
- *imm16/32*—Word (16-bit) or doubleword (32-bit) immediate.
- *imm32*—Doubleword (32-bit) immediate.
- *imm32/64*—Doubleword (32-bit) or quadword (64-bit) immediate.
- *imm64*—Quadword (64-bit) immediate.
- *mem*—An operand of unspecified size in memory.
- *mem8*—Byte (8-bit) operand in memory.
- *mem16*—Word (16-bit) operand in memory.
- *mem16/32*—Word (16-bit) or doubleword (32-bit) operand in memory.

- *mem32*—Doubleword (32-bit) operand in memory.
- *mem32/48*—Doubleword (32-bit) or 48-bit operand in memory.
- *mem48*—48-bit operand in memory.
- *mem64*—Quadword (64-bit) operand in memory.
- *mem128*—Double quadword (128-bit) operand in memory.
- *mem16:16*—Two sequential word (16-bit) operands in memory.
- *mem16:32*—A word (16-bit) operand followed by a doubleword (32-bit) operand in memory.
- *mem32real*—Single-precision (32-bit) floating-point operand in memory.
- *mem32int*—Doubleword (32-bit) integer operand in memory.
- *mem64real*—Double-precision (64-bit) floating-point operand in memory.
- *mem64int*—Quadword (64-bit) integer operand in memory.
- *mem80real*—Double-extended-precision (80-bit) floating-point operand in memory.
- *mem80dec*—80-bit packed BCD operand in memory, containing 18 4-bit BCD digits.
- *mem2env*—16-bit x87 control word or x87 status word.
- *mem14/28env*—14-byte or 28-byte x87 environment. The x87 environment consists of the x87 control word, x87 status word, x87 tag word, last non-control instruction pointer, last data pointer, and opcode of the last non-control instruction completed.
- *mem94/108env*—94-byte or 108-byte x87 environment and register stack.
- *mem512env*—512-byte environment for 128-bit media, 64-bit media, and x87 instructions.
- *mmx*—Quadword (64-bit) operand in an MMX register.
- *mmx1*—Quadword (64-bit) operand in an MMX register, specified as the left-most (first) operand in the instruction syntax.
- *mmx2*—Quadword (64-bit) operand in an MMX register, specified as the right-most (second) operand in the instruction syntax.

- *mmx/mem32*—Doubleword (32-bit) operand in an MMX register or memory.
- *mmx/mem64*—Quadword (64-bit) operand in an MMX register or memory.
- *mmx1/mem64*—Quadword (64-bit) operand in an MMX register or memory, specified as the left-most (first) operand in the instruction syntax.
- *mmx2/mem64*—Quadword (64-bit) operand in an MMX register or memory, specified as the right-most (second) operand in the instruction syntax.
- *moffset*—Memory offset of unspecified size.
- *moffset8*—Operand in memory located at the specified byte (8-bit) offset from the instruction pointer.
- *moffset16*—Operand in memory located at the specified word (16-bit) offset from the instruction pointer.
- *moffset32*—Operand in memory located at the specified doubleword (32-bit) offset from the instruction pointer.
- *pntr16:16*—Far pointer with 16-bit selector and 16-bit offset.
- *pntr16:32*—Far pointer with 16-bit selector and 32-bit offset.
- *reg*—Operand of unspecified size in a GPR register.
- *reg8*—Byte (8-bit) operand in a GPR register.
- *reg16*—Word (16-bit) operand in a GPR register.
- *reg16/32*—Word (16-bit) or doubleword (32-bit) operand in a GPR register.
- *reg32*—Doubleword (32-bit) operand in a GPR register.
- *reg64*—Quadword (64-bit) operand in a GPR register.
- *reg/mem8*—Byte (8-bit) operand in a GPR register or memory.
- *reg/mem16*—Word (16-bit) operand in a GPR register or memory.
- *reg/mem32*—Doubleword (32-bit) operand in a GPR register or memory.
- *reg/mem64*—Quadword (64-bit) operand in a GPR register or memory.
- *rel8off*—Relative address in the current code segment, in 8-bit offset range.
- *rel16off*—Relative address in the current code segment, for 16-bit operand size.

- *rel32off*—Relative address in the current code segment, for 32-bit operand size.
- *segReg* or *sReg*—Word (16-bit) operand in a segment register.
- *ST(0)*—x87 stack register 0.
- *ST(i)*—x87 stack register *i*, where *i* is between 0 and 7.
- *xmm*—Double quadword (128-bit) operand in an XMM register.
- *xmm1*—Double quadword (128-bit) operand in an XMM register, specified as the left-most (first) operand in the instruction syntax.
- *xmm2*—Double quadword (128-bit) operand in an XMM register, specified as the right-most (second) operand in the instruction syntax.
- *xmm/mem64*—Quadword (64-bit) operand in a 128-bit XMM register or memory.
- *xmm/mem128*—Double quadword (128-bit) operand in an XMM register or memory.
- *xmm1/mem128*—Double quadword (128-bit) operand in an XMM register or memory, specified as the left-most (first) operand in the instruction syntax.
- *xmm2/mem128*—Double quadword (128-bit) operand in an XMM register or memory, specified as the right-most (second) operand in the instruction syntax.

2.5.2 Opcode Syntax

In addition to the notation shown above in “Mnemonic Syntax” on page 41, the following notation indicates the size and type of operands in the syntax of an instruction opcode:

- */digit*—Indicates that the ModRM byte specifies only one register or memory (r/m) operand. The digit is specified by the ModRM reg field and is used as an instruction-opcode extension. Valid digit values range from 0 to 7.
- */r*—Indicates that the ModRM byte specifies both a register operand and a reg/mem (register or memory) operand.
- *cb, cw, cd, cp*—Specifies a code-offset value and possibly a new code-segment register value. The value following the opcode is either one byte (*cb*), two bytes (*cw*), four bytes (*cd*), or six bytes (*cp*).
- *ib, iw, id*—Specifies an immediate-operand value. The opcode determines whether the value is signed or unsigned. The value following the opcode, ModRM, or SIB byte is

either one byte (ib), two bytes (iw), or four bytes (id). Word and doubleword values start with the low-order byte.

- **+rb, +rw, +rd, +rq**—Specifies a register value that is added to the hexadecimal byte on the left, forming a one-byte opcode. The result is an instruction that operates on the register specified by the register code. Valid register-code values are shown in Table 2-2.
- **m64**—Specifies a quadword (64-bit) operand in memory.
- **+i**—Specifies an x87 floating-point stack operand, ST(i). The value is used only with x87 floating-point instructions. It is added to the hexadecimal byte on the left, forming a one-byte opcode. Valid values range from 0 to 7.

Table 2-2. +rb, +rw, +rd, and +rq Register Value

REX.B Bit ¹	Value	Specified Register			
		+rb	+rw	+rd	+rq
0 or no REX Prefix	0	AL	AX	EAX	RAX
	1	CL	CX	ECX	RCX
	2	DL	DX	EDX	RDY
	3	BL	BX	EBX	RBX
	4	AH	SP	ESP	RSP
	5	CH	BP	EBP	RBP
	6	DH	SI	ESI	RSI
	7	BH	DI	EDI	RDI
1	0	R8B	R8W	R8D	R8
	1	R9B	R9W	R9D	R9
	2	R10B	R10W	R10D	R10
	3	R11B	R11W	R11D	R11
	4	R12B	R12W	R12D	R12
	5	R13B	R13W	R13D	R13
	6	R14B	R14W	R14D	R14
	7	R15B	R15W	R15D	R15
1. See Section 1.2.7, "REX Prefixes", on page 14.					

2.5.3 Pseudocode Definitions

Pseudocode examples are given for the actions of several complex instructions in Section 2.6, “General-Purpose Instruction Reference”, on page 57, Section 2.10, “System Instruction Reference”, on page 1047, and elsewhere throughout this manual (for example, see the CALL instruction on page 87). The following definitions apply to all such pseudocode examples:

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Basic Definitions
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// All comments start with these double slashes.

REAL_MODE      = (cr0.pe=0)
PROTECTED_MODE = ((cr0.pe=1) && (rflags.vm=0))
VIRTUAL_MODE   = ((cr0.pe=1) && (rflags.vm=1))
LEGACY_MODE    = (efer.lma=0)
LONG_MODE      = (efer.lma=1)
64BIT_MODE     = ((efer.lma=1) && (cs.L=1)) && cs.d=0
COMPATIBILITY_MODE = (efer.lma=1) && (cs.L=0)
PAGING_ENABLED = (cr0.pg=1)
ALIGNMENT_CHECK_ENABLED = ((cr0.am=1) && (eflags.ac=1) && (cpl=3))
CPL            = the current privilege level (0-3)
OPERAND_SIZE   = 16, 32, or 64 (depending on current code and 66h/rex prefixes)
ADDRESS_SIZE   = 16, 32, or 64 (depending on current code and 67h prefixes)
STACK_SIZE     = 16, 32, or 64 (depending on current code and SS.attr.B)

old_RIP        = RIP at the start of current instruction
old_RSP        = RSP at the start of current instruction
old_RFLAGS     = RFLAGS at the start of the instruction
old_CS         = CS selector at the start of current instruction
old_DS         = DS selector at the start of current instruction
old_ES         = ES selector at the start of current instruction
old_FS         = FS selector at the start of current instruction
old_GS         = GS selector at the start of current instruction
old_SS        = SS selector at the start of current instruction

RIP            = the current RIP register
RSP            = the current RSP register
RBP            = the current RBP register
RFLAGS         = the current RFLAGS register
NEXT_RIP       = RIP at start of next instruction

CS             = the current CS descriptor, including the subfields:
                 sel base limit attr
SS             = the current SS descriptor, including the subfields:
                 sel base limit attr

SRC            = the instruction's Source operand

```

```

DEST                = the instruction's Destination operand

temp_*              // 64-bit temporary register

temp_*_desc         // temporary descriptor, with subfields:
                    //     if it points to a block of memory: sel base limit attr
                    //     if it's a gate descriptor: sel offset segment attr

NULL = 0x0000      // null selector is all zeros

// V,Z,A,S are integer variables, assigned a value when an instruction begins
// executing (they can be assigned a different value in the middle of an
// instruction, if needed)

V = 2 if OPERAND_SIZE=16
   4 if OPERAND_SIZE=32
   8 if OPERAND_SIZE=64

Z = 2 if OPERAND_SIZE=16
   4 if OPERAND_SIZE=32
   4 if OPERAND_SIZE=64

A = 2 if ADDRESS_SIZE=16
   4 if ADDRESS_SIZE=32
   8 if ADDRESS_SIZE=64

S = 2 if STACK_SIZE=16
   4 if STACK_SIZE=32
   8 if STACK_SIZE=64

/////////////////////////////////////////////////////////////////
// Bit Range Inside a Register
/////////////////////////////////////////////////////////////////

temp_data.[X:Y]      // Bit X through Y in temp_data, with the other bits
                    // in the register masked off.

/////////////////////////////////////////////////////////////////
// Moving Data From One Register To Another
/////////////////////////////////////////////////////////////////

temp_dest.b = temp_src // 1-byte move (copies lower 8 bits of temp_src to
                        // temp_dest, preserving the upper 56 bits of temp_dest)
temp_dest.w = temp_src // 2-byte move (copies lower 16 bits of temp_src to
                        // temp_dest, preserving the upper 48 bits of temp_dest)
temp_dest.d = temp_src // 4-byte move (copies lower 32 bits of temp_src to
                        // temp_dest, and zeros out the upper 32 bits of temp_dest)
temp_dest.q = temp_src // 8-byte move (copies all 64 bits of temp_src to
                        // temp_dest)

```

```
temp_dest.v = temp_src    // 2-byte move if V=2,
                          // 4-byte move if V=4,
                          // 8-byte move if V=8
```

```
temp_dest.z = temp_src    // 2-byte move if Z=2,
                          // 4-byte move if Z=4
```

```
temp_dest.a = temp_src    // 2-byte move if A=2,
                          // 4-byte move if A=4,
                          // 8-byte move if A=8
```

```
temp_dest.s = temp_src    // 2-byte move if S=2,
                          // 4-byte move if S=4,
                          // 8-byte move if S=8
```

```
/////////////////////////////////////////////////////////////////
// Bitwise Operations
/////////////////////////////////////////////////////////////////
```

```
temp = a AND b
temp = a OR b
temp = a XOR b
temp = NOT a
temp = a SHL b
temp = a SHR b
```

```
/////////////////////////////////////////////////////////////////
// Logical Operations
/////////////////////////////////////////////////////////////////
```

```
IF (F00 && BAR)
IF (F00 || BAR)
IF (F00 = BAR)
IF (F00 != BAR)
IF (F00 > BAR)
IF (F00 < BAR)
IF (F00 >= BAR)
IF (F00 <= BAR)
```

```
/////////////////////////////////////////////////////////////////
// IF-THEN-ELSE
/////////////////////////////////////////////////////////////////
```

```
IF (F00)
    ...
```



```

IF (F00)
    ...
ELSIF (BAR)
    ...
ELSE
    ...

IF ((F00 && BAR) || (CONE && HEAD))
    ...

////////////////////////////////////
// Exceptions
////////////////////////////////////

EXCEPTION [#GP(0)]      // error code in parenthesis
EXCEPTION [#UD]         // if no error code

possible exception types:

#DE      // Divide-By-Zero-Error Exception (Vector 0)
#DB      // Debug Exception (Vector 1)
#BP      // INT3 Breakpoint Exception (Vector 3)
#OF      // INTO Overflow Exception (Vector 4)
#BR      // Bound-Range Exception (Vector 5)
#UD      // Invalid-Opcode Exception (Vector 6)
#NM      // Device-Not-Available Exception (Vector 7)
#DF      // Double-Fault Exception (Vector 8)
#TS      // Invalid-TSS Exception (Vector 10)
#NP      // Segment-Not-Present Exception (Vector 11)
#SS      // Stack Exception (Vector 12)
#GP      // General-Protection Exception (Vector 13)
#PF      // Page-Fault Exception (Vector 14)
#MF      // x87 Floating-Point Exception-Pending (Vector 16)
#AC      // Alignment-Check Exception (Vector 17)
#MC      // Machine-Check Exception (Vector 18)
#XF      // SIMD Floating-Point Exception (Vector 19)

#SX      // security exception
#SHUTDOWN // shutdown

////////////////////////////////////
// READ_MEM
// General memory read. This zero-extends the data to 64 bits and returns it.
////////////////////////////////////

usage:
    temp = READ_MEM.x [seg:offset] // where x is one of {v, z, b, w, d, q}
                                     // and denotes the size of the memory read

```

definition:

```

IF ((seg F0xFFFC) = NULL)      // GP fault for using a null segment to
                                // reference memory
    EXCEPTION [#GP(0)]

IF ((seg=CS) || (seg=DS) || (seg=ES) || (seg=FS) || (seg=GS))
    // CS,DS,ES,FS,GS check for segment limit or canonical
    IF ((!64BIT_MODE) && (offset is outside seg's limit))
        EXCEPTION [#GP(0)]
        // #GP fault for segment limit violation in non-64-bit mode
    IF ((64BIT_MODE) && (offset is non-canonical))
        EXCEPTION [#GP(0)]
        // #GP fault for non-canonical address in 64-bit mode
ELIF (seg=SS)    // SS checks for segment limit or canonical
    IF ((!64BIT_MODE) && (offset is outside seg's limit))
        EXCEPTION [#SS(0)]
        // stack fault for segment limit violation in non-64-bit mode
    IF ((64BIT_MODE) && (offset is non-canonical))
        EXCEPTION [#SS(0)]
        // stack fault for non-canonical address in 64-bit mode
ELSE // ((seg=GDT) || (seg=LDT) || (seg=IDT) || (seg=TSS))
    // GDT,LDT,IDT,TSS check for segment limit and canonical
    IF (offset > seg.limit)
        EXCEPTION [#GP(0)] // #GP fault for segment limit violation
                            // in all modes
    IF ((LONG_MODE) && (offset is non-canonical))
        EXCEPTION [#GP(0)] // #GP fault for non-canonical address in long mode

IF ((ALIGNMENT_CHECK_ENABLED) && (offset misaligned, considering its size and
    alignment))
    EXCEPTION [#AC(0)]

IF ((64_bit_mode) && ((seg=CS) || (seg=DS) || (seg=ES) || (seg=SS)))
    temp_linear = offset
ELSE
    temp_linear = seg.base + offset

IF ((PAGING_ENABLED) && (virtual-to-physical translation for temp_linear
    results in a page-protection violation))
    EXCEPTION [#PF(error_code)] // page fault for page-protection violation
                                // (U/S violation, Reserved bit violation)

IF ((PAGING_ENABLED) && (temp_linear is on a not-present page))
    EXCEPTION [#PF(error_code)] // page fault for not-present page

temp_data = memory [temp_linear].x // zero-extends the data to 64
                                    // bits, and saves it in temp_data

RETURN (temp_data)                // return the zero-extended data

```

```

////////////////////////////////////
// WRITE_MEM // General memory write
////////////////////////////////////

```

usage:

```

WRITE_MEM.x [seg:offset] = temp.x    // where <X> is one of these:
                                     // {V, Z, B, W, D, Q} and denotes the
                                     // size of the memory write

```

definition:

```

IF ((seg & 0xFFFFC)= NULL)          // GP fault for using a null segment to refer-
    ence memory
    EXCEPTION [#GP(0)]

IF (seg isn't writable)              // GP fault for writing to a read-only segment
    EXCEPTION [#GP(0)]

IF ((seg=CS) || (seg=DS) || (seg=ES) || (seg=FS) || (seg=GS))
    // CS,DS,ES,FS,GS check for segment limit or canonical
    IF ((!64BIT_MODE) && (offset is outside seg's limit))
        EXCEPTION [#GP(0)]
        // #GP fault for segment limit violation in non-64-bit mode
    IF ((64BIT_MODE) && (offset is non-canonical))
        EXCEPTION [#GP(0)]
        // #GP fault for non-canonical address in 64-bit mode
ELIF (seg=SS)                        // SS checks for segment limit or canonical
    IF ((!64BIT_MODE) && (offset is outside seg's limit))
        EXCEPTION [#SS(0)]
        // stack fault for segment limit violation in non-64-bit mode
    IF ((64BIT_MODE) && (offset is non-canonical))
        EXCEPTION [#SS(0)]
        // stack fault for non-canonical address in 64-bit mode
ELSE // ((seg=GDT) || (seg=LDT) || (seg=IDT) || (seg=TSS))
    // GDT,LDT,IDT,TSS check for segment limit and canonical
    IF (offset > seg.limit)
        EXCEPTION [#GP(0)]
        // #GP fault for segment limit violation in all modes
    IF ((LONG_MODE) && (offset is non-canonical))
        EXCEPTION [#GP(0)]
        // #GP fault for non-canonical address in long mode

IF ((ALIGNMENT_CHECK_ENABLED) && (offset is misaligned, considering its size
    and alignment))
    EXCEPTION [#AC(0)]

IF ((64_bit_mode) && ((seg=CS) || (seg=DS) || (seg=ES) || (seg=SS))
    temp_linear = offset
ELSE
    temp_linear = seg.base + offset

```

```

IF ((PAGING_ENABLED) && (the virtual-to-physical translation for
temp_linear results in a page-protection violation))
{
    EXCEPTION [#PF(error_code)]
        // page fault for page-protection violation
        // (U/S violation, Reserved bit violation)
}

IF ((PAGING_ENABLED) && (temp_linear is on a not-present page))
    EXCEPTION [#PF(error_code)]    // page fault for not-present page

memory [temp_linear].x = temp.x    // write the bytes to memory

////////////////////////////////////
// PUSH // Write data to the stack
////////////////////////////////////

usage:
    PUSH.x temp    // where x is one of these: {v, z, b, w, d, q} and
                    // denotes the size of the push

definition:

    WRITE_MEM.x [SS:RSP.s - X] = temp.x    // write to the stack
    RSP.s = RSP - X    // point rsp to the data just written

////////////////////////////////////
// POP // Read data from the stack, zero-extend it to 64 bits
////////////////////////////////////

usage:
    POP.x temp    // where x is one of these: {v, z, b, w, d, q} and
                    // denotes the size of the pop

definition:

    temp = READ_MEM.x [SS:RSP.s]    // read from the stack
    RSP.s = RSP + X    // point rsp above the data just written

////////////////////////////////////
// READ_DESCRIPTOR // Read 8-byte descriptor from GDT/LDT, return the descriptor
////////////////////////////////////

usage:
    temp_descriptor = READ_DESCRIPTOR (selector, chktype)

    // cs_chk    used for far call and far jump

```

```
// clg_chk    used when reading CS for far call or far jump through call gate
// ss_chk     used when reading SS
// iret_chk   used when reading CS for IRET or RETF
// intcs_chk  used when reading the CS for interrupts and exceptions
```

definition:

```
temp_offset = selector AND 0xffff8      // upper 13 bits give an offset
                                         // in the descriptor table

IF (selector.TI = 0)                    // read 8 bytes from the gdt, split it into
                                         // (base,limit,attr) if the type bits
    temp_desc = READ_MEM.q [gdt:temp_offset]
                                         // indicate a block of memory, or split it into
                                         // (segment,offset,attr) if the type bits indicate
                                         // a gate, and save the result in temp_desc
ELSE
    temp_desc = READ_MEM.q [ldt:temp_offset]
                                         // read 8 bytes from the ldt, split it into
                                         // (base,limit,attr) if the type bits
                                         // indicate a block of memory, or split it into
                                         // (segment,offset,attr) if the type
                                         // bits indicate a gate, and save the result in
                                         // temp_desc

IF (selector.rpl or temp_desc.attr.dpl is illegal for the current mode/cpl)
    EXCEPTION [#GP(selector)]

IF (temp_desc.attr.type is illegal for the current mode/chktype)
    EXCEPTION [#GP(selector)]

IF (temp_desc.attr.p=0)
    EXCEPTION [#NP(selector)]

RETURN (temp_desc)
```

```
/////////////////////////////////////////////////////////////////
// READ_IDT // Read an 8-byte descriptor from the IDT, return the descriptor
/////////////////////////////////////////////////////////////////
```

usage:

```
temp_idt_desc = READ_IDT (vector)
                // "vector" is the interrupt vector number
```

definition:

```
IF (LONG_MODE)        // long-mode idt descriptors are 16 bytes long
    temp_offset = vector*16
ELSE // (LEGACY_MODE) legacy-protected-mode idt descriptors are 8 bytes long
    temp_offset = vector*8
```

```

temp_desc = READ_MEM.q [idt:temp_offset]
                // read 8 bytes from the idt, split it into
                // (segment,offset,attr), and save it in temp_desc

IF (temp_desc.attr.dpl is illegal for the current mode/cpl)
    // exception, with error code that indicates this idt gate
    EXCEPTION [#GP(vector*8+2)]

IF (temp_desc.attr.type is illegal for the current mode)
    // exception, with error code that indicates this idt gate
    EXCEPTION [#GP(vector*8+2)]

IF (temp_desc.attr.p=0)
    EXCEPTION [#NP(vector*8+2)]
                // segment-not-present exception, with an error code that
                // indicates this idt gate

RETURN (temp_desc)

////////////////////////////////////
// READ_INNER_LEVEL_STACK_POINTER
// Read a new stack pointer (rsp or ss:esp) from the tss
////////////////////////////////////

usage:
    temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER (new_cpl, ist_index)

definition:

IF (LONG_MODE)
    IF (ist_index>0)
        // if IST is selected, read an ISTn stack pointer from the tss
        temp_RSP = READ_MEM.q [tss:ist_index*8+28]
    ELSE // (ist_index=0)
        // otherwise read an RSPn stack pointer from the tss
        temp_RSP = READ_MEM.q [tss:new_cpl*8+4]
        temp_SS_desc.sel = NULL + new_cpl
                // in long mode, changing to lower cpl sets SS.sel to
                // NULL+new_cpl
    ELSE // (LEGACY_MODE)
    {
        temp_RSP = READ_MEM.d [tss:new_cpl*4]                // read ESPn from the tss
        temp_sel = READ_MEM.d [tss:new_cpl*8+8]              // read SSn from the tss
        temp_SS_desc = READ_DESCRIPTOR (temp_sel, ss_chk)
    }

return (temp_RSP:temp_SS_desc)

```

```

////////////////////////////////////////////////////////////////
// READ_BIT_ARRAY // Read 1 bit from a bit array in memory
////////////////////////////////////////////////////////////////

```

usage:

```
temp_value = READ_BIT_ARRAY ([mem], bit_number)
```

definition:

```
temp_BYTE = READ_MEM.b [mem + (bit_number SHR 3)]
               // read the byte containing the bit

temp_BIT = temp_BYTE SHR (bit_number & 7)
               // shift the requested bit position into bit 0

return (temp_BIT & 0x01)    // return '0' or '1'

```


2.6 General-Purpose Instruction Reference

This section describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by the general-purpose instructions. General-purpose instructions are used in basic software execution. Most of these instructions load, store, or operate on data located in the general-purpose registers (GPRs), in memory, or in both. The remaining instructions are used to alter the sequential flow of the program by branching to other locations within the program, or to entirely different programs. The instructions within this category do not operate on any other register set.

General-purpose instructions are supported in all hardware implementations of the x86-64 architecture, except that the following general-purpose instructions are implemented only if their associated CUID function bit is set:

- CMPXCHG8B, indicated by bit 8 of CUID standard function 1 and extended function 8000_0001h.
- CMOVcc (conditional moves), indicated by bit 15 of CUID standard function 1 and extended function 8000_0001h.
- Long Mode instructions, indicated by bit 29 of CUID extended function 8000_0001h.

The general-purpose instructions can be used in legacy mode or long mode. Compilation of general-purpose programs for execution in 64-bit mode offers three primary advantages: access to the eight extended, 64-bit general-purpose registers (for a register set consisting of GPR0–GPR15), access to the 64-bit virtual address space, and access to the RIP-relative addressing mode.

For further information about the general-purpose instructions and register resources, see:

- Volume 1: Section 3, “General-Purpose Programming”, on page 27.
- Volume 3: Section 2.3, “Summary of Registers and Data Types”, on page 28.
- Volume 3: Section 2.5, “Notation”, on page 41.
- Volume 3: Section 1.2, “Instruction Prefixes”, on page 3.
- Volume 3: Appendix A, “General-Purpose Instructions in 64-Bit Mode”, on page 1163. In particular, see Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163.

AAA**ASCII Adjust After Addition**

The AAA instruction is used after an ADD instruction has added two unpacked decimal digits in ASCII code. This makes it possible to add ASCII numbers without masking off the upper nibble ‘3’. After executing the AAA instruction, the AL register contains the correct decimal digit of the result, and the AH register is incremented by one if a decimal carry occurred during addition. The AF and CF flags are set if a decimal carry occurred during addition (i.e., if the value of the AL register was greater than 9 or if the AF flag was set), otherwise the AF and CF flags are cleared. The other arithmetic flags (OF, SF, ZF, PF) are undefined.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
AAA	37	Creates an unpacked BCD number. (Invalid in 64-bit mode.)

Related Instructions

AAD, AAM, AAS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	1/0	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

AAD

ASCII Adjust Before Division

The AAD instruction converts two unpacked BCD digits in AL and AH to binary. Executing this instruction before the DIV instruction allows division of ASCII numbers. The AAD instruction sets the value in the AL register to $(AL + (10 * AH))$, and then clears the AH register to 00H. The value in the AX register is then equal to the binary equivalent of the original unpacked two-digit (base 10) number in registers AH and AL.

Before the AAD instruction is executed, the ASCII tag of 3 must be masked from the upper nibble of AH and AL.

The AAD instruction mnemonic always adjusts base-10 results. To carry adjust to any other base, the instruction must be coded directly in binary. The base is specified as in immediate byte value (imm8) suffixed onto opcode D5h. These values are 08h for octal, 0Ah for decimal, and 0Ch for duodecimals. The AAD mnemonic is interpreted by all assemblers to mean adjust ASCII (base 10) values. To adjust values in another number base, the instruction must be hand coded in machine code (D5 ib).

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
AAD	D5 0A	Adjusts two BCD digits in AL and AH. (Invalid in 64-bit mode.)
None	D5 <i>ib</i>	Adjust two BCD digits to the immediate byte base. (Invalid in 64-bit mode.)

Related Instructions

AAA, AAM, AAS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				1/0	1/0	U	1/0	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

AAM

ASCII Adjust After Multiply

Converts the value in the AX register from binary to unpacked BCD. AAM adjusts only AL. Any digits greater than 9 are stored in AH. The AAM instruction is used after the MUL instruction has multiplied two unpacked BCD (base ten) numbers.

The AAM instruction mnemonic always adjusts base ten results. To carry adjust to any other base, the instruction must be coded directly in binary. The base is specified as an immediate byte value (imm8) suffixed onto opcode D4h. Example values are 08h for octal, 0Ah for decimal, and 0Ch for duodecimals. The AAM mnemonic is interpreted by all assemblers to mean adjust to ASCII (base 10) values. To adjust to values in another number base, the instruction must be hand coded in machine code (D4 ib).

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
AAM	D4 0A	Creates a pair of unpacked BCD values in AH and AL. (Invalid in 64-bit mode.)
None	D4 <i>ib</i>	Creates a pair of unpacked values to the immediate byte base. (Invalid in 64-bit mode.)

Related Instructions

AAA, AAD, AAS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				1/0	1/0	U	1/0	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.
Divide by zero, #DE	X	X	X	8-bit immediate value is 0.

AAS**ASCII Adjust After Subtraction**

Converts the binary value in AL to one digit unpacked BCD result. The value in AH is decremented if the carry flag is set. If no decimal carry occurred, the CF and AF flags are cleared, and the AH register is unchanged. In either case, the AL register is left with its top nibble cleared to 0.

This instruction is used after a binary subtraction of one unpacked BCD value from another, leaving a byte result in the AL register. The AAA instruction then adjusts the contents of the AL register to contain the correct 1-digit unpacked BCD result.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
AAS	3F	Creates an unpacked BCD number from the contents of the AL register. (Invalid in 64-bit mode.)

Related Instructions

AAA, AAD, AAM.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	1/0	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

ADC Add with Carry

Adds the destination operand, the source operand, and the carry (CF) flag and stores the result in the destination. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) The value of the CF flag represents a carry from a previous addition. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

This instruction does not distinguish between signed or unsigned operands. The processor evaluates the result for either data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The value of SF flag indicates the sign of a signed result.

The ADC instruction is usually executed after an ADD as part of a multibyte or multiword addition.

The LOCK prefix can be used with forms of the ADC instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
ADC AL, <i>imm8</i>	14 <i>ib</i>	Add <i>imm8</i> to AL with carry.
ADC AX, <i>imm16</i>	15 <i>iv</i>	Add <i>imm16</i> to AX with carry.
ADC EAX, <i>imm32</i>	15 <i>id</i>	Add <i>imm32</i> to EAX with carry.
ADC RAX, <i>imm32</i>	15 <i>id</i>	Add <i>imm32</i> to RAX with carry.
ADC <i>reg/mem8</i> , <i>imm8</i>	80 /2 <i>ib</i>	Add <i>imm8</i> to byte register or memory with carry.
ADC <i>reg/mem16</i> , <i>imm16</i>	81 /2 <i>iv</i>	Add <i>imm16</i> to word register or memory with carry.
ADC <i>reg/mem32</i> , <i>imm32</i>	81 /2 <i>id</i>	Add <i>imm32</i> to double register or memory with carry flag.
ADC <i>reg/mem64</i> , <i>imm32</i>	81 /2 <i>id</i>	Add <i>imm32</i> to quadword register or memory with carry flag.
ADC <i>reg/mem16</i> , <i>imm8</i>	83 /2 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem16</i> with sign-extension based on CF.
ADC <i>reg/mem32</i> , <i>imm8</i>	83 /2 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem32</i> with sign-extension based on CF.
ADC <i>reg/mem64</i> , <i>imm8</i>	83 /2 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem64</i> with sign-extension based on CF.

Mnemonic	Opcode	Description
<i>ADC reg/mem8, reg8</i>	10 /r	Add contents of byte memory or byte register to contents of byte register with carry.
<i>ADC reg/mem16, reg16</i>	11 /r	Add contents of word memory or word register to contents of word register with carry.
<i>ADC reg/mem32, reg32</i>	11 /r	Add contents of double memory or double register to contents of double register with carry.
<i>ADC reg/mem64, reg64</i>	11 /r	Add contents of quadword memory or quadword register to contents of quadword register with carry.
<i>ADC reg8, reg/mem8</i>	12 /r	Add contents of byte register or memory to byte register with carry.
<i>ADC reg16, reg/mem16</i>	13 /r	Add contents of word register or memory to word register with carry.
<i>ADC reg32, reg/mem32</i>	13 /r	Add contents of doubleword register or memory to doubleword register with sign-extended carry flag.
<i>ADC reg64, reg/mem64</i>	13 /r	Add contents of quadword register or memory to quadword register with sign-extended carry flag.

Related Instructions

ADD, SUB.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

ADD Signed or Unsigned Add

Adds source operand to destination operand and places the result in destination. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

Mnemonic	Opcode	Description
ADD AL, <i>imm8</i>	04 <i>ib</i>	Add <i>imm8</i> to AL.
ADD AX, <i>imm16</i>	05 <i>iw</i>	Add <i>imm16</i> to AX.
ADD EAX, <i>imm32</i>	05 <i>id</i>	Add <i>imm32</i> to EAX.
ADD RAX, <i>imm32</i>	05 <i>id</i>	Add <i>imm32</i> to RAX.
ADD <i>reg/mem8</i> , <i>imm8</i>	80 /0 <i>ib</i>	Add <i>imm8</i> to <i>reg/mem8</i> .
ADD <i>reg/mem16</i> , <i>imm16</i>	81 /0 <i>iw</i>	Add <i>imm16</i> to <i>reg/mem16</i>
ADD <i>reg/mem32</i> , <i>imm32</i>	81 /0 <i>id</i>	Add <i>imm32</i> to <i>reg/mem32</i> .
ADD <i>reg/mem64</i> , <i>imm32</i>	81 /0 <i>id</i>	Add <i>imm32</i> to <i>reg/mem64</i> .
ADD <i>reg/mem16</i> , <i>imm8</i>	83 /0 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem8</i> .
ADD <i>reg/mem32</i> , <i>imm8</i>	83 /0 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem32</i> .
ADD <i>reg/mem64</i> , <i>imm8</i>	83 /0 <i>ib</i>	Add sign-extended <i>imm8</i> to <i>reg/mem64</i> .
ADD <i>reg/mem8</i> , <i>reg8</i>	00 / <i>r</i>	Add <i>reg8</i> to <i>reg/mem8</i> .
ADD <i>reg/mem16</i> , <i>reg16</i>	01 / <i>r</i>	Add <i>reg16</i> to <i>reg/mem16</i> .
ADD <i>reg/mem32</i> , <i>reg32</i>	01 / <i>r</i>	Add <i>reg32</i> to <i>reg/mem32</i> .
ADD <i>reg/mem64</i> , <i>reg64</i>	01 / <i>r</i>	Add <i>reg64</i> to <i>reg/mem64</i> .
ADD <i>reg8</i> , <i>reg/mem8</i>	02 / <i>r</i>	Add <i>reg/mem8</i> to <i>reg8</i> .
ADD <i>reg16</i> , <i>reg/mem16</i>	03 / <i>r</i>	Add <i>reg/mem16</i> to <i>reg16</i> .
ADD <i>reg32</i> , <i>reg/mem32</i>	03 / <i>r</i>	Add <i>reg/mem32</i> to <i>reg32</i> .
ADD <i>reg64</i> , <i>reg/mem64</i>	03 / <i>r</i>	Add <i>reg/mem64</i> to <i>reg64</i> .

In the case of addition of unsigned numbers, the status of CF, ZF, and AF must be considered. The most important of these flags is CF. It becomes 1 if there is carry from

D7 out in 8-bit (D0–D7) operations, from D15 out in 16-bit (D0–D15) operations, or from D31 out in 32-bit (D0–D31) operations.

In the case of addition of signed numbers, the status of OF, ZF, and SF must be noted. Special attention should be given to the overflow flag (OF) since this indicates if there is an error in the result of the addition. There are two rules for setting OF in signed number operation. The overflow flag is set to 1:

- If there is a carry from D6 to D7 and no carry from D7 out in an 8-bit operation.
- If there is a carry from D14 to D15 and no carry from D15 out in a 16-bit operation.
- If there is a carry from D30 to D31 and no carry from D31 out in a 32-bit operation.
- If there is a carry from D62 to D63 and no carry from D63 out in a 64-bit operation.
- If there is a carry from D7 out and no carry from D6 to D7 in an 8-bit operation.
- If there is a carry from D15 out but no carry from D14 to D15 in a 16-bit operation.
- If there is a carry from D31 out but no carry from D30 to D31 in a 32-bit operation.
- If there is a carry from D63 out but no carry from D62 to D63 in a 64-bit operation.

Notice that if there is a carry both from D7 out and from D6 to D7, then OF = 0 in 8-bit operations. In 16-bit operations, OF = 0 if there is both a carry out from D15 and a carry from D14 to D15. And in 32-bit operations, OF = 0 if there is both a carry out from D31 and a carry from D30 to D32.

The LOCK prefix can be used with forms of the ADD instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Related Instructions

SUB, ADC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

AND Logical AND

Performs a bitwise AND operation on the destination and source operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.)

Each bit of the result of the AND instruction is set to 1 if both corresponding bits of the operands are set; otherwise, it is cleared to 0. (See truth table below.)

Mnemonic	Opcode	Description
AND AL, <i>imm8</i>	24 <i>ib</i>	AND the contents of the AL register with <i>imm8</i> .
AND AX, <i>imm16</i>	25 <i>iw</i>	AND the contents of the AX register with <i>imm16</i> .
AND EAX, <i>imm32</i>	25 <i>id</i>	AND the contents of the EAX register with <i>imm32</i> .
AND RAX, <i>imm32</i>	25 <i>id</i>	AND the contents of the RAX register with <i>imm32</i> .
AND <i>reg/mem8</i> , <i>imm8</i>	80 /4 <i>ib</i>	AND the contents of <i>reg/mem8</i> with <i>imm8</i> .
AND <i>reg/mem16</i> , <i>imm16</i>	81 /4 <i>iw</i>	AND the contents of <i>reg/mem16</i> with <i>imm16</i> .
AND <i>reg/mem32</i> , <i>imm32</i>	81 /4 <i>id</i>	AND the contents of <i>reg/mem32</i> with <i>imm32</i> .
AND <i>reg/mem64</i> , <i>imm32</i>	81 /4 <i>id</i>	AND the contents of <i>reg/mem64</i> with <i>imm32</i> .
AND <i>reg/mem16</i> , <i>imm8</i>	83 /4 <i>ib</i>	AND the contents of <i>reg/mem16</i> with <i>imm8</i> .
AND <i>reg/mem32</i> , <i>imm8</i>	83 /4 <i>ib</i>	AND the contents of <i>reg/mem32</i> with <i>imm8</i> .
AND <i>reg/mem64</i> , <i>imm8</i>	83 /4 <i>ib</i>	AND the contents of <i>reg/mem64</i> with <i>imm8</i> .
AND <i>reg/mem8</i> , <i>reg8</i>	20 / <i>r</i>	AND the contents of an 8-bit register or memory operand with the contents of an 8-bit register.
AND <i>reg/mem16</i> , <i>reg16</i>	21 / <i>r</i>	AND the contents of a 16-bit register or memory operand with the contents of a 16-bit register.
AND <i>reg/mem32</i> , <i>reg32</i>	21 / <i>r</i>	AND the contents of a 32-bit register or memory operand with the contents of a 32-bit register.
AND <i>reg/mem64</i> , <i>reg64</i>	21 / <i>r</i>	AND the contents of a 64-bit register or memory operand with the contents of a 64-bit register.
AND <i>reg8</i> , <i>reg/mem8</i>	22 / <i>r</i>	AND the contents of an 8-bit register with the contents of an 8-bit memory or register operand.
AND <i>reg16</i> , <i>reg/mem16</i>	23 / <i>r</i>	AND the contents of a 16-bit register with the contents of a 16-bit memory or register operand.

Mnemonic	Opcode	Description
AND <i>reg32, reg/mem32</i>	23 /r	AND the contents of a 32-bit register with the contents of a 32-bit memory or register operand.
AND <i>reg64, reg/mem64</i>	23 /r	AND the contents of a 64-bit register with the contents of a 64-bit memory or register operand.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

The LOCK prefix can be used with forms of the AND instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Related Instructions

TEST, OR, NOT, NEG.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				1/0	1/0	U	1/0	0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BOUND**Check Array Bounds**

Checks whether an array index in the destination operand is within the bounds of an array specified in the source operand. The array index is a signed integer specified in a register and the array operand is a memory location containing a pair of signed doubleword-integers (when the operand-size attribute is 32) or a pair of signed word-integers (when the operand-size attribute is 16). The first doubleword (or word) is the lower bound of the array and the second doubleword (or word) is the upper bound of the array. The array index must be greater than or equal to the lower bound and less than or equal to the upper bound plus the operand size in bytes. If the index is not within bounds, a BOUND range exceeded exception (#BR) is signaled. (When a this exception is generated, the saved return instruction pointer points to the BOUND instruction.)

The bounds limit data structure (two words or doublewords containing the lower and upper limits of the array) is usually placed just before the array itself, making the limits addressable via a constant offset from the beginning of the array. This practice avoids extra bus cycles to obtain the effective address of the array bounds, since the address of the array is already in a register.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
BOUND <i>reg32, mem32&mem32</i>	62/r	Tests whether a 32-bit array index is within the bounds specified by the two 32-bit values in <i>mem32&mem32</i> . (Invalid in 64-bit mode.)
BOUND <i>reg16, mem16&mem16</i>	62/r	Tests whether a 16-bit array index is within the bounds specified by the two 16-bit values in <i>mem16&mem16</i> . (Invalid in 64-bit mode.)

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Bound range, #BR	X	X	X	The bound range was exceeded.
Invalid opcode, #UD	X	X	X	The second operand is not a memory location.
			X	Software is executing in 64-bit mode.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BSF**Bit Scan Forward**

Searches a source operand for the least-significant set bit. If a set bit is found, the zero flag is cleared and the bit index of the first set bit is loaded into the destination operand. The bit index is zero-based. The least-significant bit is bit zero.

The source operand can be a register or a memory location; the destination operand is a register. The bit index is an unsigned offset from bit 0 of the source operand. If the source operand contains 0 and $ZF = 1$, the contents of the destination operand is undefined.

Mnemonic	Opcode	Description
<i>BSF reg16, reg/mem16</i>	0F BC	Bit scan forward on the contents of <i>reg16</i> .
<i>BSF reg32, reg/mem32</i>	0F BC	Bit scan forward on the contents of <i>reg32</i> .
<i>BSF reg64, reg/mem64</i>	0F BC	Bit scan forward on the contents of <i>reg64</i> .

Related Instructions

BSR.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	1/0	U	U	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BSR**Bit Scan Reverse**

Searches a source operand for the most-significant set bit. If a set bit is found, the bit index is stored in the destination operand. If no set bit is found, the zero flag is set and the contents of the destination are undefined. The bit index is zero-based.

Mnemonic	Opcode	Description
BSR <i>reg16, reg/mem16</i>	0F BD	Bit scan reverse on the contents of <i>reg/mem16</i> .
BSR <i>reg32, reg/mem32</i>	0F BD	Bit scan reverse on the contents of <i>reg/mem32</i> .
BSR <i>reg64, reg/mem64</i>	0F BD	Bit scan reverse on the contents of <i>reg/mem64</i> .

Related Instructions

BSF.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	1/0	U	U	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BSWAP

Byte Swap

Converts the contents of a register from little endian to big endian or vice versa. (Swaps the byte order of a register.) In a doubleword, bits 7:0 are exchanged with bits 31:24, and bits 15:8 are exchanged with bits 23:16. In a quadword, bits 7:0 are exchanged with bits 63:56, bits 15:8 with bits 55:48, bits 23:16 with bits 47:40, and bits 31:24 with bits 39:32. A subsequent use of the BSWAP instruction with the same operand restores the original value of the operand.

BSWAP can be used to speed up binary-coded decimal (BCD) arithmetic operations because it allows processing of four BCD digits stored in a register. The `htonl()` and `ntohl()` C-library functions can be implemented using the BSWAP instruction.

The result of applying the BSWAP instruction to a 16-bit register is undefined. To swap bytes of a 16-bit register, use the XCHG instruction.

Mnemonic	Opcode	Description
BSWAP <i>reg32</i>	D9 F0	Reverse the byte order of <i>reg32</i> .
BSWAP <i>reg64</i>	D9 F0	Reverse the byte order of <i>reg64</i> .

Related Instructions

CMPXCHG, XCHG.

rFLAGS Affected

None.

Exceptions

None.

BT**Bit Test**

Copies a specified bit (second operand) from the first operand (bit-base) to the carry flag (CF) of the rFLAGS register. The bit-base may be a register or memory location. The bit is specified by either the (zero-based) index in the source register or by an 8-bit immediate value.

If the bit base operand is a register, the instruction takes the modulo 16, 32, 64 (depending on the register size) of the bit offset operand, allowing any bit position to be selected in a 16-, 32- or 64-bit register.

If the bit-base operand specifies a memory location, it represents the address of the byte in memory that contains the bit-base (bit 0 of the specified byte) of the bit string (see Figure 3-2). The offset operand then selects a bit position within the range -2^{63} to $2^{63}-1$ for a register offset and 0 to 63 for an immediate offset.

When accessing a bit in memory, the processor accesses 2-, 4-, 8- bytes starting from the memory address for a 16-, 32- or 64-bit operand size, using by the following relationship:

Effective Address + (NumBytes_i * (BitOffset DIV NumBits_i*8))

When using this bit addressing mechanism, software should avoid referencing areas of memory close to address space holes, such as references to memory-mapped I/O registers. Instead, software should use the MOV instructions to load from or store to these addresses, and use the register form of these instructions to manipulate the data.

Mnemonic	Opcode	Description
BT <i>reg/mem16, reg16</i>	0F A3	Set the carry flag to the value of the selected bit.
BT <i>reg/mem32, reg32</i>	0F A3	Set the carry flag to the value of the selected bit.
BT <i>reg/mem64, reg64</i>	0F A3	Set the carry flag to the value of the selected bit.
BT <i>reg/mem16, imm8</i>	0F BA /4 <i>ib</i>	Set the carry flag to the value of the selected bit.
BT <i>reg/mem32, imm8</i>	0F BA /4 <i>ib</i>	Set the carry flag to the value of the selected bit.
BT <i>reg/mem64, imm8</i>	0F BA /4 <i>ib</i>	Set the carry flag to the value of the selected bit.

Related Instructions

BTC, BTR, BTS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BTC**Bit Test and Complement**

Copies a bit specified by the second operand from the first operand, which may be a register or memory location, to the carry flag (CF) of the rFLAGS register and toggles the bit in the first operand. This instruction uses zero-based indexing to specify the bit to test.

If the bit base operand specifies a register, the instruction takes the modulo 16 or 32 (depending on the register size) of the bit offset operand, allowing any bit position to be selected in a 16- or 32-bit register, respectively.

This instruction is useful for the implementation of semaphores in concurrent operating systems. In such applications, this instruction should be preceded by the LOCK prefix. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
<i>BTC reg/mem16, reg16</i>	0F BB	Set the carry flag to the value of the selected bit and complement the bit in the bit string.
<i>BTC reg/mem32, reg32</i>	0F BB	Set the carry flag to the value of the selected bit and complement the bit in the bit string.
<i>BTC reg/mem64, reg64</i>	0F BB	Set the carry flag to the value of the selected bit and complement the bit in the bit string.
<i>BTC reg/mem16, imm8</i>	0F BA /7 ib	Set the carry flag to the value of the selected bit and complement the bit in the bit string.
<i>BTC reg/mem32, imm8</i>	0F BA /7 ib	Set the carry flag to the value of the selected bit and complement the bit in the bit string.
<i>BTC reg/mem64, imm8</i>	0F BA /7 ib	Set the carry flag to the value of the selected bit and complement the bit in the bit string.

Related Instructions

BT, BTR, BTS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BTR**Bit Test and Reset**

Copies a bit specified by the second operand from the first operand, which may be a register or memory location, to the carry flag (CF) of the rFLAGS register and clears the target bit in the operand. The target bit can also be specified by an 8-bit immediate value.

If the bit base operand specifies a register, the instruction takes the modulo 16, 32 or 64 (depending on the register size) of the bit offset operand, allowing any bit position to be selected in a 16-bit, 32-bit, or 64-bit register, respectively.

This instruction is useful for the implementation of semaphores in concurrent operating systems. In such applications, this instruction should be preceded by the LOCK prefix. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
<i>BTR reg/mem16, reg16</i>	0F B3	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.
<i>BTR reg/mem32, reg32</i>	0F B3	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.
<i>BTR reg/mem64, reg64</i>	0F B3	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.
<i>BTR reg/mem16, imm8</i>	0F BA /6 <i>ib</i>	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.
<i>BTR reg/mem32, imm8</i>	0F BA /6 <i>ib</i>	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.
<i>BTR reg/mem64, imm8</i>	0F BA /6 <i>ib</i>	Set the carry flag to the value of the selected bit and clear the bit in the bit string to zero.

Related Instructions

BT, BTC, BTR, BTS.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

BTS**Bit Test and Set**

Copies a specified target bit-base specified by the second operand from the source operand, which may be a register or memory location, to the carry flag (CF) of the rFLAGS register and sets the target bit in the destination to 1. The index bit can also be specified by an 8-bit immediate value. The specified bit is specified by a zero-based index.

If the bit base operand specifies a register, the instruction takes the modulo 16 or 32 (depending on the register size) of the bit offset operand, allowing any bit position to be selected in a 16- or 32-bit register, respectively.

This instruction is useful for the implementation of semaphores in concurrent operating systems. In such applications, this instruction should be preceded by the LOCK prefix. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
<i>BTS reg/mem16, reg16</i>	0F AB	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.
<i>BTS reg/mem32, reg32</i>	0F AB	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.
<i>BTS reg/mem64, reg64</i>	0F AB	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.
<i>BTS reg/mem16, imm8</i>	0F BA /5 <i>ib</i>	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.
<i>BTS reg/mem32, imm8</i>	0F BA /5 <i>ib</i>	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.
<i>BTS reg/mem64, imm8</i>	0F BA /5 <i>ib</i>	Sets the carry flag to the value of the selected bit and sets the bit in the bit string to 1.

Related Instructions

BT, BTC, BTR.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CALL Procedure Call

Transfers control to a procedure. There are four types of calls: near, far, inter-privilege-level, task switch.

Mnemonic	Opcode	Description
CALL <i>rel16off</i>	E8 <i>iw</i>	Near call to code in the current code segment at 16-bit address specified by <i>rel16off</i> . (Cannot be encoded in 64-bit mode.)
CALL <i>rel32off</i>	E8 <i>id</i>	Near call to code in the current code segment at 32-bit address specified by <i>rel32off</i> .
CALL <i>reg/mem16</i>	FF /2	Near call to address specified in the memory location specified by <i>reg/mem16</i> . (Cannot be encoded in 64-bit mode.)
CALL <i>reg/mem32</i>	FF /2	Near call to address specified in the memory location specified by <i>reg/mem32</i> .
CALL <i>reg/mem64</i>	FF /2	Near call to address specified in the memory location specified by <i>reg/mem64</i> .
CALL <i>pntr16:16</i>	9A <i>cd</i>	Absolute far call. (Invalid in 64-bit mode.)
CALL <i>pntr16:32</i>	9A <i>cp</i>	Absolute far call. (Invalid in 64-bit mode.)
CALL <i>mem16:16</i>	FF /3	Absolute indirect far call.
CALL <i>mem16:32</i>	FF /3	Absolute indirect far call.

A procedure accessed by a near CALL is located in the same code segment as the CALL instruction. The content of the rIP register (which is the address of the instruction after the CALL) is pushed onto the stack and rSP is decremented by the appropriate number of bytes. Then rIP is loaded with the new value, which is the offset of the near procedure.

At the end of the procedure, RET is used to return control to the instruction after the call. When RET is executed, the rIP is popped off the stack, which returns control to the instruction after the CALL. Direct, register indirect, or memory indirect addressing can be used to code the address of the called near procedure.

For near calls in 64-bit mode, the operand size defaults to 64 bits, the E8 opcode results in $RIP = RIP + 32\text{-bit displacement sign-extended to 64 bits}$, and the FF /2 opcode results in $RIP = 64\text{-bit offset from register or memory}$. (The 16-bit displacement size cannot be encoded, because operand size is fixed at 64 bits.)

When calling a far procedure (the procedure is in a different segment from the CALL instruction), the rSP is decremented after the CS:rIP of the instruction following the CALL is pushed onto the stack. CS:rIP is then loaded with the segment and offset address of the called procedure. When pushing CS:rIP onto the stack, CS is pushed first and then rIP. When the RETF is executed, CS and rIP are restored from the stack and execution continues with the instruction following the CALL.

For far calls in 64-bit mode, the 9A opcode is invalid, and the FF /3 opcode does the following:

- For 32-bit operand size, if the selector points to a gate, then RIP = zero-extended 32-bit offset from the gate, else RIP = zero-extended 32-bit offset from the far pointer referenced in instruction.
- For 64-bit operand size, if the selector points to a gate, then RIP = 64-bit offset from the gate, else RIP = zero-extended 32-bit offset from the far pointer referenced in instruction.

For details about control-flow instructions, see Section 3.7, “Control Transfers”, on page 95 of Volume 1, and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.

Action

```
// Far calls (CALLF)
// See Section 2.5.3, “Pseudocode Definitions”, on page 46.
```

```
CALLF_START:
```

```
IF (REAL_MODE)
    CALLF_REAL_OR_VIRTUAL
ELSIF (PROTECTED_MODE)
    CALLF_PROTECTED
ELSE // (VIRTUAL_MODE)
    CALLF_REAL_OR_VIRTUAL
```

```
CALLF_REAL_OR_VIRTUAL:
```

```
    IF (OPCODE = callf [mem])
    {
        temp_RIP = READ_MEM.z [mem]
        temp_CS  = READ_MEM.w [mem+Z]
    }
    ELSE // (OPCODE = callf direct)
    {
        temp_RIP = z-sized offset specified in the instruction,
                  zero-extended to 64 bits
        temp_CS  = selector specified in the instruction
    }
```



```

PUSH.v old_CS
PUSH.v next_RIP

IF (temp_RIP>CS.limit)
    EXCEPTION [#GP(0)]

CS.sel = temp_CS
CS.base = temp_CS SHL 4
RIP = temp_RIP
EXIT

```

CALLF_PROTECTED:

```

IF (OPCODE = callf [mem])
{
    temp_offset = READ_MEM.z [mem]
    temp_sel     = READ_MEM.w [mem+Z]
}
ELSE // (OPCODE = callf direct)
    IF (64BIT_MODE)
        EXCEPTION [#UD]          // 'callf direct' is illegal in 64-bit mode
    {
        temp_offset = z-sized offset specified in the instruction,
                        zero-extended to 64 bits
        temp_sel     = selector specified in the instruction
    }

temp_desc = READ_DESCRIPTOR (temp_sel, cs_chk)

IF (temp_desc.attr.type = 'available_tss')
    TASK_SWITCH // using temp_sel as the target tss selector
ELSIF (temp_desc.attr.type = 'taskgate')
    TASK_SWITCH // using the tss selector in the task gate as the target tss
ELSIF (temp_desc.attr.type = 'code')
    // if the selector refers to a code descriptor, then
    // the offset we read is the target RIP
{
    temp_RIP = temp_offset
    CS = temp_desc
    PUSH.v old_CS
    PUSH.v next_RIP
    IF ((!64BIT_MODE) && (temp_RIP > CS.limit))
        EXCEPTION [#GP(0)] // temp_RIP can't be non-canonical because
                            // it's a 16- or 32-bit number, zero-extended
                            // to 64 bits

    RIP = temp_RIP
    EXIT
}
ELSE // (temp_desc.attr.type = 'callgate')
    // if the selector refers to a call gate, then

```

```

    // the target CS and RIP both come from the call gate

IF (LONG_MODE)
    // the size of the gate controls the size of the stack pushes
    V=8-byte
    // long mode only uses 64-bit call gates, force 8-byte opsize
ELSIF (temp_desc.attr.type = 'callgate32')
    V=4-byte
    // legacy mode, using a 32-bit call-gate, force 4-byte opsize
ELSE
    // (temp_desc.attr.type = 'callgate16')
    V=2-byte
    // legacy mode, using a 16-bit call-gate, force 2-byte opsize

temp_RIP = temp_desc.offset

IF (LONG_MODE)
    // in long mode, we need to read the 2nd half of a
    // 16-byte call-gate from the gdt/ldt, to get the upper
    // 32 bits of the target RIP
    temp_upper = READ_MEM.q [temp_sel+8]
    IF (temp_upper.[44:40] != 0)
        EXCEPTION [#GP(0)]
        // make sure the extended attribute bits are all zero
    temp_RIP = temp_RIP + (temp_upper SHL 32)
    // concatenate both halves of RIP

CS = READ_DESCRIPTOR (temp_desc.segment, clg_chk)

temp_CPL = CS.sel.rpl
IF (CPL=temp_CPL)
{
    PUSH.v old_CS
    PUSH.v next_RIP
}
IF ((64BIT_MODE) && (temp_RIP is non-canonical)
    || (!64BIT_MODE) && (temp_RIP > CS.limit))
{
    EXCEPTION #GP(0)
}

RIP = temp_RIP
EXIT

ELSE // (CPL != temp_CPL), changing privilege
{
    CPL = temp_CPL
    temp_ist = 0 // call-far doesn't use ist pointers
    temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER (CPL, temp_ist)

    RSP.q = temp_RSP
    SS = temp_SS_desc
    PUSH.v old_SS // SS on following pushes use ss.sel as error code

```

```

    PUSH.v old_RSP
    IF (LEGACY_MODE)      // legacy-mode call gates have a param_count field
        temp_PARAM_COUNT = temp_desc.attr.param_count

    FOR (I=temp_PARAM_COUNT; I>0; I--)
    {
        temp_DATA = READ_MEM.v [SS:old_RSP+I*V]
        PUSH.v temp_DATA
    }

    PUSH.v old_CS
    PUSH.v next_RIP
    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
    {
        EXCEPTION [#GP(0)]
    }
    RIP = temp_RIP
    EXIT

```

Related Instructions

RET.

rFLAGS Affected

None, except if task switch occurs, in which case all flags are affected.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	The 9A opcode of this instruction is not recognized in 64-bit mode.
Invalid TSS, #TS (selector)			X	The target stack segment and ESP point to a location that is past the end of the TSS.
			X	The selector for the target stack segment is null.
			X	The new stack-segment selector in the TSS contains an RPL that is not equal to the code-segment descriptor's DPL.
			X	The target stack-segment descriptor contains a DPL that is not equal to that of the code-segment descriptor.
			X	The target stack segment is not a writable data segment.
			X	The index in the stack-segment selector is beyond the descriptor table's limits.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Segment not present, #NP (selector)			X	A segment, call gate, task gate, or TSS is not present.
Stack, #SS			X	The stack segment limit was exceeded when a stack switch occurred.
			X	The memory operand effective address exceeded the stack segment limit.
Stack, #SS (selector)			X	The stack segment limit was exceeded when a stack switch occurred.
			X	As part of a stack switch, the SS register was loaded but the pointed-to segment was not present.
			X	When the stack switch occurred, there was not enough space in the stack segment for the return address, the parameters, or the stack-segment pointer.
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
	X	X	X	The offset for the destination operand is outside the boundaries of the target code segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
			X	The the destination operand's segment selector is null.
			X	The gate contains a code-segment selector that is null.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP (selector)			X	The index for the code segment, gate, or TSS selector is beyond the limits of the descriptor table.
			X	The segment descriptor is not a nonconforming-code segment, conforming-code segment, task gate, call gate, or TSS.
			X	The segment selector's RPL is greater than the CPL, or the nonconforming-code segment's DPL is not equal to the CPL.
			X	A conforming-code segment's DPL is greater than the CPL.
			X	A call-gate, task-gate, or TSS descriptor's DPL is less than the CPL, or it is less than the RPL of the call-gate, task-gate, or TSS segment selector.
			X	The descriptor for a call gate's segment selector does not specify it as a code segment.
			X	A call gate's segment selector is outside the limits of the descriptor table.
			X	The code-segment DPL that is specified by a call gate is greater than the CPL.
			X	The TI bit in the TSS's segment selector indicates a local table.
			X	The segment descriptor for the TSS indicates that the TSS is not available or busy.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CBW**Convert Byte to Word**

Copies the value of the sign flag (SF) to all bits of AX. Used to convert a signed byte in AL into a signed word to avoid the overflow problem in signed number arithmetic.

Mnemonic	Opcode	Description
CBW	98	Sign extend AL into AX.

Related Instructions

CDQ, CWD, CWDE, CDQE.

rFLAGS Affected

None.

Exceptions

None.

CDQ

Convert Doubleword to Quadword

Doubles the size of the operand in the EAX register by sign extension and stores the result in registers EDX:EAX. The CDQ instruction copies the sign (bit 31) of the value in the EAX register into every bit position in the EDX register.

The CDQ instruction can be used to produce a quadword dividend from a double-word before doubleword division.

The CWD and CDQ mnemonics reference the same opcode. Both mnemonics are provided for semantic reasons.

Mnemonic	Opcode	Description
CDQ	99	Sign extend EAX into EDX:EAX.

Related Instructions

CBW, CWD, MOVSX, MOVZX.

rFLAGS Affected

None.

Exceptions

None.

CDQE**Convert Doubleword to Quadword (64-bit mode)**

This mnemonic is meaningful only in 64-bit mode.

Extends the doubleword sign bit in EAX to fill a quadword in RAX. Used to convert a signed doubleword into a signed quadword to avoid overflow in signed number arithmetic.

Mnemonic	Opcode	Description
CDQE	98	Sign extend EAX into RAX.

Related Instructions

CBW, CWDE, CWD, CDQ, CQO.

rFLAGS Affected

None.

Exceptions

None.

CLC Clear Carry Flag

Resets the carry flag (CF) in rFLAGS to zero.

Mnemonic	Opcode	Description
CLC	F8	Clear the carry flag to zero.

Related Instructions

STC, CMC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
																0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

None.

CLD Clear Direction Flag

Resets the direction flag (DF) in the rFLAGS register to zero. In string instructions, if DF = 0, the pointers (index registers ESI and/or EDI) are incremented with each execution of the instruction. If DF = 1, the pointers are decremented. The CLD is used before string instructions to make the pointers increment.

Mnemonic	Opcode	Description
CLD	FC	Clear the direction flag to zero.

Action

```
directionFlag = 0;
```

Related Instructions

INSx, LODSx, MOVSx, OUTSx, SCASx, STD, STOSx.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
									0							
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

None.

CLFLUSH

Cache Line Invalidate

The CLFLUSH instruction invalidates the cache line specified by the *mem8* linear-address operand. All levels of the cache hierarchy—internal caches and external caches—are checked, and the cache line is invalidated in every cache in which it is found. If a cache contains a dirty copy of the cache line (that is, the cache line is in the *modified* or *owned* MOESI state), the line is written back to memory before it is invalidated. CLFLUSH sets the cache-line MOESI state to *invalid*.

The physical address corresponding to the CLFLUSH linear-address operand is also checked against the processor's write-combining buffers. If data intended for that physical address is held in the write-combining buffer, the entire contents of the buffer are written to memory. This occurs even though the data is not cached in the cache hierarchy. In a multiprocessor system, only the write-combining buffers on the processor that executed the CLFLUSH instruction are checked.

CLFLUSH is weakly-ordered with respect to other instructions that operate on memory. Speculative loads initiated by the processor, or specified explicitly using cache-prefetch instructions, can be reordered around a CLFLUSH. Such reordering can cause freshly-loaded cache lines to be flushed unintentionally. This situation can be avoided *only* by using the MFENCE instruction to force strong-ordering of CLFLUSH with respect to other memory operations. LFENCE, SFENCE, and serializing instructions are *not* ordered with respect to CLFLUSH.

CLFLUSH behaves like a load instruction with respect to setting the page-table accessed and dirty bits. That is, CLFLUSH sets the page-table accessed bit to 1, but does not modify the page-table dirty bit.

The CLFLUSH bit returned by CPUID standard function 1 indicates whether the processor supports the CLFLUSH instruction. CLFLUSH can be executed at any privilege level, but invalidates the cache line only if access is permitted by the segment-protection and page-protection settings for the specified address. Cache lines have the same segment protection as one byte reads (or loads); if the execute only bit is set, execute only code segments are allowed.

Mnemonic	Opcode	Description
CFLUSH <i>mem8</i>	OF AE /7	Invalidates cache line at memory location <i>mem8</i> .

Related Instructions

INVD, WBINVD

rFLAGS Affected

None.

Exceptions

Exception (vector)	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The CLFLUSH Instruction bit (bit 19) in CPUID standard function 1 is cleared to 0.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

CMC**Complement Carry Flag**

Changes CF from 0 to 1 or from 1 to 0.

Mnemonic	Opcode	Description
CMC	F5	Complement the carry flag.

Action

```
carryFlag = not(carryFlag);
```

Related Instructions

CLC, STC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
																1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

None.

CMOVcc Conditional Move

These instructions conditionally move a 16-, 32-, or 64-bit value in memory or a general-purpose register into a register, depending upon the settings of condition flags in the rFLAGS register.

Most assemblers provide instruction mnemonics with A (above) and B (below) tags to supply the semantics for manipulating unsigned integers. Those with G (greater than) and L (less than) tags are provided for dealing with signed integers. Thus, many opcodes may be represented by synonymous mnemonics. For example, the CMOVA instruction is synonymous with the CMOVNBE instruction.

Mnemonic	Opcode	Description
CMOVA <i>reg16, reg/mem16</i> CMOVNBE <i>reg16, reg/mem16</i>	0F 47 /r	Move if carry flag and zero flag are both zero.
CMOVA <i>reg32, reg/mem32</i> CMOVNBE <i>reg32, reg/mem32</i>	0F 47 /r	Move if carry flag and zero flag are both zero.
CMOVA <i>reg64, reg/mem64</i> CMOVNBE <i>reg64, reg/mem64</i>	0F 47 /r	Move if carry flag and zero flag are both zero.
CMOVAE <i>reg16, reg/mem16</i> CMOVNB <i>reg16, reg/mem16</i> CMOVNC <i>reg16, reg/mem16</i>	0F 43 /r	Move if carry flag is zero.
CMOVAE <i>reg32, reg/mem32</i> CMOVNB <i>reg32, reg/mem32</i> CMOVNC <i>reg32, reg/mem32</i>	0F 43 /r	Move if carry flag is zero.
CMOVAE <i>reg64, reg/mem64</i> CMOVNB <i>reg64, reg/mem64</i> CMOVNC <i>reg64, reg/mem64</i>	0F 43 /r	Move if carry flag is zero.
CMOVB <i>reg16, reg/mem16</i> CMOVC <i>reg16, reg/mem16</i> CMOVNAE <i>reg16, reg/mem16</i>	0F 42 /r	Move if carry flag is 1 (below).
CMOVB <i>reg32, reg/mem32</i> CMOVC <i>reg32, reg/mem32</i> CMOVNAE <i>reg32, reg/mem32</i>	0F 42 /r	Move if carry flag is 1 (below).
CMOVB <i>reg64, reg/mem64</i> CMOVC <i>reg64, reg/mem64</i> CMOVNAE <i>reg64, reg/mem64</i>	0F 42 /r	Move if carry flag is 1 (below).
CMOVBE <i>reg16, reg/mem16</i> CMOVNA <i>reg16, reg/mem16</i>	0F 46 /r	Move if carry flag or zero flag is 1 (below or equal).

CMOVBE <i>reg32, reg/mem32</i> CMOVNA <i>reg32, reg/mem32</i>	0F 46/r	Move if carry flag or zero flag is 1 (below or equal).
CMOVBE <i>reg64, reg/mem64</i> CMOVNA <i>reg64, reg/mem64</i>	0F 46/r	Move if carry flag or zero flag is 1 (below or equal).
CMOVE <i>reg16, reg/mem16</i> CMOVZ <i>reg16, reg/mem16</i>	0F 44/r	Move if zero flag is 1 (equal).
CMOVE <i>reg32, reg/mem32</i> CMOVZ <i>reg32, reg/mem32</i>	0F 44/r	Move if zero flag is 1 (equal/zero).
CMOVE <i>reg64, reg/mem64</i> CMOVZ <i>reg64, reg/mem64</i>	0F 44/r	Move if zero flag is 1 (equal/zero).
CMOVG <i>reg16, reg/mem16</i> CMOVNLE <i>reg16, reg/mem16</i>	0F 4F/r	Move if zero flag is zero and the sign flag is equal to the overflow flag (greater/not less or equal).
CMOVG <i>reg32, reg/mem32</i> CMOVNLE <i>reg32, reg/mem32</i>	0F 4F/r	Move if zero flag is zero and the sign flag is equal to the overflow flag (greater/not less or equal).
CMOVG <i>reg64, reg/mem64</i> CMOVNLE <i>reg64, reg/mem64</i>	0F 4F/r	Move if zero flag is zero and the sign flag is equal to the overflow flag (greater/not less or equal).
CMOVGE <i>reg16, reg/mem16</i> CMOVNL <i>reg16, reg/mem16</i>	0F 4D/r	Move if the sign flag is equal to the overflow flag.
CMOVGE <i>reg32, reg/mem32</i> CMOVNL <i>reg32, reg/mem32</i>	0F 4D/r	Move if the sign flag is equal to the overflow flag.
CMOVGE <i>reg64, reg/mem64</i> CMOVNL <i>reg64, reg/mem64</i>	0F 4D/r	Move if the sign flag is equal to the overflow flag.
CMOVL <i>reg16, reg/mem16</i> CMOVNGE <i>reg16, reg/mem16</i>	0F 4C/r	Move if the sign flag is not equal to the overflow flag.
CMOVL <i>reg32, reg/mem32</i> CMOVNGE <i>reg32, reg/mem32</i>	0F 4C/r	Move if the sign flag is not equal to the overflow flag.
CMOVL <i>reg64, reg/mem64</i> CMOVNGE <i>reg64, reg/mem64</i>	0F 4C/r	Move if the sign flag is not equal to the overflow flag.
CMOVLE <i>reg16, reg/mem16</i> CMOVNG <i>reg16, reg/mem16</i>	0F 4E/r	Move if the zero flag is 1 or the sign flag is not equal to the overflow flag.
CMOVLE <i>reg32, reg/mem32</i> CMOVNG <i>reg32, reg/mem32</i>	0F 4E/r	Move if the zero flag is 1 or the sign flag is not equal to the overflow flag.
CMOVLE <i>reg64, reg/mem64</i> CMOVNG <i>reg64, reg/mem64</i>	0F 4E/r	Move if the zero flag is 1 or the sign flag is not equal to the overflow flag.
CMOVNE <i>reg16, reg/mem16</i>	0F 45/r	Move if zero flag is zero.
CMOVNE <i>reg32, reg/mem32</i>	0F 45/r	Move if zero flag is zero.
CMOVNE <i>reg64, reg/mem64</i>	0F 45/r	Move if zero flag is zero.
CMOVNO <i>reg16, reg/mem16</i>	0F 41/r	Move if the overflow flag is 0.

CMOVNO <i>reg32, reg/mem32</i>	0F 41 /r	Move if the overflow flag is 0.
CMOVNO <i>reg64, reg/mem64</i>	0F 41 /r	Move if the overflow flag is 0.
CMOVNP <i>reg16, reg/mem16</i>	0F 4B /r	Move if the parity flag is 0.
CMOVNP <i>reg32, reg/mem32</i>	0F 4B /r	Move if the parity flag is 0.
CMOVNP <i>reg64, reg/mem64</i>	0F 4B /r	Move if the parity flag is 0.
CMOVNS <i>reg16, reg/mem16</i>	0F 49 /r	Move if the sign flag is 0.
CMOVNS <i>reg32, reg/mem32</i>	0F 49 /r	Move if the sign flag is 0.
CMOVNS <i>reg64, reg/mem64</i>	0F 49 /r	Move if the sign flag is 0.
CMOVNZ <i>reg16, reg/mem16</i>	0F 45 /r	Move if the zero flag is not 0.
CMOVNZ <i>reg32, reg/mem32</i>	0F 45 /r	Move if the zero flag is not 0.
CMOVNZ <i>reg64, reg/mem64</i>	0F 45 /r	Move if the zero flag is not 0.
CMOVO <i>reg16, reg/mem16</i>	0F 40 /r	Move if the overflow flag is 1.
CMOVO <i>reg32, reg/mem32</i>	0F 40 /r	Move if the overflow flag is 1.
CMOVO <i>reg64, reg/mem64</i>	0F 40 /r	Move if the overflow flag is 1.
CMOVP <i>reg16, reg/mem16</i>	0F 4A /r	Move if the parity flag is 1.
CMOVP <i>reg32, reg/mem32</i>	0F 4A /r	Move if the parity flag is 1.
CMOVP <i>reg64, reg/mem64</i>	0F 4A /r	Move if the parity flag is 1.
CMOVPE <i>reg16, reg/mem16</i>	0F 4A /r	Move if the parity flag is 1.
CMOVPE <i>reg32, reg/mem32</i>	0F 4A /r	Move if the parity flag is 1.
CMOVPE <i>reg64, reg/mem64</i>	0F 4A /r	Move if the parity flag is 1.
CMOVPO <i>reg16, reg/mem16</i>	0F 4B /r	Move if the parity flag is 0.
CMOVPO <i>reg32, reg/mem32</i>	0F 4B /r	Move if the parity flag is 0.
CMOVPO <i>reg64, reg/mem64</i>	0F 4B /r	Move if the parity flag is 0.
CMOVS <i>reg16, reg/mem16</i>	0F 48 /r	Move if the sign flag is 1.
CMOVS <i>reg32, reg/mem32</i>	0F 48 /r	Move if the sign flag is 1.
CMOVS <i>reg64, reg/mem64</i>	0F 48 /r	Move if the sign flag is 1.

CMOVcc instructions perform the same task as **MOV** but work conditionally, depending on the state of the status flags in the **rFLAGS** register. If the condition is not satisfied, the instruction has no effect and control is passed to the next instruction. The mnemonics of **CMOVcc** instructions denote the condition that must be satisfied. Several mnemonics are often used for one opcode to make instructions names easier to remember. For example, **CMOVE** (conditional move if equal) and **CMOVZ** (conditional move if zero) are aliases and denote the same instruction with the opcode 0F 44h.

Support for CMOVcc instructions depends on the processor implementation. To find out if a processor is able to perform CMOVcc instructions, use the CPUID instruction to determine whether bit 15 of CPUID standard function 1 and extended function 8000_0001h is set to 1.

Related Instructions

MOV, MOVD, MOVQ, MOVS, MOVSB, MOVSW, MOVSD, MOVSX, MOVZX.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The Conditional Move Instructions bit (bit 15) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CMP**Compare**

Compares the contents of the first register or memory operand with the contents of second operand, which can be a register, memory address, or immediate operand, and sets the status flags in the rFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and setting the status flags in the same manner as the SUB instruction, except that the contents of the destination register remains unchanged. When an immediate value is used as an operand, it is sign-extended to the length of the first operand.

The CMP instruction is typically used in conjunction with a conditional jump (Jcc), conditional move (CMOVcc), or conditional SETcc instruction. The condition codes used by the Jcc, CMOVcc, and SETcc instructions may be based on the results of a CMP instruction. Appendix D, “Instruction Effects on rFLAGS”, on page 1247 shows how instructions affect the status flags, which can be used as condition codes.

Mnemonic	Opcode	Description
CMP AL, <i>imm8</i>	3C <i>ib</i>	Compare an 8-bit immediate value with the contents of the AL register.
CMP AX, <i>imm16</i>	3D <i>iw</i>	Compare a 16-bit immediate value with the contents of the AX register.
CMP EAX, <i>imm32</i>	3D <i>id</i>	Compare a 32-bit immediate value with the contents of the EAX register.
CMP RAX, <i>imm32</i>	3D <i>id</i>	Compare a 32-bit immediate value with the contents of the RAX register.
CMP <i>reg/mem8</i> , <i>imm8</i>	80 /7 <i>ib</i>	Compare an 8-bit immediate value with the contents of an 8-bit register or memory operand.
CMP <i>reg/mem16</i> , <i>imm16</i>	81 /7 <i>iw</i>	Compare a 16-bit immediate value with the contents of a 16-bit register or memory operand.
CMP <i>reg/mem32</i> , <i>imm32</i>	81 /7 <i>id</i>	Compare a 32-bit immediate value with the contents of a 32-bit register or memory operand.
CMP <i>reg/mem64</i> , <i>imm32</i>	81 /7 <i>id</i>	Compare a 32-bit immediate value with the contents of a 64-bit register or memory operand.
CMP <i>reg/mem16</i> , <i>imm8</i>	83 /7 <i>ib</i>	Compare the contents of a 16-bit register or memory operand with an 8-bit immediate value.
CMP <i>reg/mem32</i> , <i>imm8</i>	83 /7 <i>ib</i>	Compare the contents of a 32-bit register or memory operand with an 8-bit immediate value.
CMP <i>reg/mem64</i> , <i>imm8</i>	83 /7 <i>ib</i>	Compare the contents of a 64-bit register or memory operand with an 8-bit immediate value.

Mnemonic	Opcode	Description
<i>CMP reg/mem8, reg8</i>	38/r	Compare the contents of an 8-bit register or memory operand with the contents of an 8-bit register.
<i>CMP reg/mem16, reg16</i>	39/r	Compare the contents of a 16-bit register or memory operand with the contents of a 16-bit register.
<i>CMP reg/mem32, reg32</i>	39/r	Compare the contents of a 32-bit register or memory operand with the contents of a 32-bit register.
<i>CMP reg/mem64, reg64</i>	39/r	Compare the contents of a 64-bit register or memory operand with the contents of a 64-bit register.
<i>CMP reg8, reg/mem8</i>	3A/r	Compare the contents of an 8-bit register with the contents of an 8-bit register or memory operand.
<i>CMP reg16, reg/mem16</i>	3B/r	Compare the contents of a 16-bit register with the contents of a 16-bit register or memory operand.
<i>CMP reg32, reg/mem32</i>	3B/r	Compare the contents of a 32-bit register with the contents of a 32-bit register or memory operand.
<i>CMP reg64, reg/mem64</i>	3B/r	Compare the contents of a 64-bit register with the contents of a 64-bit register or memory operand.

When comparing unsigned operands, flag settings are as follows:

Operands	CF	ZF
dest > source	0	0
dest = source	0	1
dest < source	1	0

When comparing signed operands, flag settings are as follows:

Operands	ZF	OF
dest > source	0	SF
dest = source	1	Undefined
dest < source	Undefined	NOT SF

Related Instructions

SUB, CMPS, CMPSB, CMPSD, CMPSW.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CMPS Compare Strings

CMPSB

CMPSW

CMPSD

CMPSQ

Compares two strings a byte, word, doubleword, or quadword at a time. The source and destination operands must be the same size. The first source operand is addressed at *seg*:*rSI*. The value of *seg* defaults to the DS segment, but may be overridden by a segment prefix. The second source operand is always addressed at ES:*rDI*. The CMPS instruction sets the bits of the rFLAGS register according to the outcome of the comparison. The comparison is performed by subtracting the source operand from the destination and setting rFLAGS values as if a SUB were performed.

After the comparison, the rSI and rDI registers are incremented or decremented according to the setting of the DF flag in the rFLAGS register. If DF = 0, rSI and rDI are incremented by 1, 2, 4 or 8. If DF = 1, the pointers are decremented by 1, 2, 4, or 8.

The CMPSB instruction compares two strings a byte at a time. DS:SI is used to address the first operand; ES:DI is used to address the second.

The CMPSW instruction compares two strings a word at a time. DS:SI is used to address the first operand; ES:DI is used to address the second.

The CMPSD instruction compares two strings a doubleword at a time. EDS:ESI is used to address the first operand; ES:EDI is used to address the second. This CMPSD instruction should not be confused with the same-mnemonic CMPSD (compare scalar double-precision floating-point) instruction in the 128-bit media instruction set. Assemblers can distinguish the instructions by the number and type of operands.

The CMPSQ instruction compares two strings a quadword at a time. RDS:RSI is used to address the first operand; ES:RDI is used to address the second.

Mnemonic	Opcode	Description
CMPS m8, m8	A6	Compares the byte at DS:SI with the byte at ES:DI and sets flags in RFLAGS to reflect the result.
CMPS m16, m16	A7	Compares the word at DS:SI with the word at ES:DI and sets flags in RFLAGS to reflect the result.

CMPS m32, m32	A7	Compares the doubleword at DS:ESI with the doubleword at ES:EDI and sets flags in RFLAGS to reflect the result.
CMPS m64, m64	A7	Compares the quadword at DS:RSI with the quadword at ES:RDI and sets flags in RFLAGS to reflect the result.
CMPSB	A6	Compares the byte at DS:SI with the byte at ES:DI and sets flags in RFLAGS to reflect the result.
CMPSW	A7	Compares the word at DS:SI with the word at ES:DI and sets flags in RFLAGS to reflect the result.
CMPSD	A7	Compares the doubleword at DS:ESI with the doubleword at ES:EDI and sets flags in RFLAGS to reflect the result.
CMPSQ	A7	Compares the quadword at DS:RSI with the quadword at ES:RDI and sets flags in RFLAGS to reflect the result.

The forms of CMPSx that have explicit operands are equivalent to the forms with implicit operands (in fact, they have the same opcodes). The explicit forms, however, only allow specification of the operand size and an override of the default DS segment used to access the [rSI] operand. The addresses of these explicit operands are otherwise ignored, and instead, the implicit operands—*seg*:[rSI] and ES:[rDI]—are always used.

The REPE or REPZ prefixes (they are synonyms) and the REPNE or REPNZ prefixes (they are synonyms) can be used with the CMPS instruction. For details about the REP prefix, see “Repeat Prefixes” on page 10. If a CMPSx instruction is followed by a conditional jump instruction like JL, the jump is performed if the value of the *seg*:[rSI] operand is less than the ES:[rDI] operand. This allows lexicographical comparisons of string or array elements. A CMPSx instruction can also be put inside a loop controlled by the LOOPcc instruction.

Related Instructions

CMP, CMPSB, CMPSW, CMPSD.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CMPXCHG Compare and Exchange

Compares a value in the AL, AX, EAX, or RAX register with another register or a memory location. If the two values are equal, the source operand value is loaded into the first operand. Otherwise, the contents of the first operand is loaded into the AL, AX, EAX, or RAX register.

The ZF flag is set to 1 if the values in the destination operand and register AL, AX, or EAX are equal; otherwise ZF is cleared to 0.

The LOCK prefix can be used with forms of the CMPXCHG instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
CMPXCHG <i>reg/mem8</i> , AL	0F B0 /r	Compares the contents of the AL register to the contents of an 8-bit destination register or memory operand. If the two are equal, set the zero flag to 1 and load the contents of the AL register into the register or memory operand. If they are not equal, load the contents of the register or memory operand into AL.
CMPXCHG <i>reg/mem16</i> , AX	0F B1 /r	Compares the contents of the AX register to the contents of an 16-bit destination register or memory operand. If the two are equal, set the zero flag to 1 and load the contents of the AX register into the register or memory operand. If they are not equal, load the contents of the register or memory operand into AX.
CMPXCHG <i>reg/mem32</i> , EAX	0F B1 /r	Compares the contents of the EAX register to the contents of an 32-bit destination register or memory operand. If the two are equal, set the zero flag to 1 and load the contents of the EAX register into the register or memory operand. If they are not equal, load the contents of the register or memory operand into EAX.
CMPXCHG <i>reg/mem64</i> , RAX	0F B1 /r	Compares the contents of the RAX register to the contents of a 64-bit destination register or memory operand. If the two are equal, set the zero flag to 1 and load the contents of the RAX register into the register or memory operand. If they are not equal, load the contents of the register or memory operand into RAX.

Related Instructions**CMPXCHG8B**

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:

Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

CMPXCHG8B

Compare and Exchange Eight Bytes

Compares a 64-bit value in the EDX:EAX register with a 64-bit memory operand. If the values are equal, the contents of the ECX:EBX register is loaded into the specified memory operand and the zero flag (ZF) is set to 1. Otherwise, the operand is loaded into the EDX:EAX register and ZF is cleared to 0.

The LOCK prefix can be used with forms of the CMPXCHG8B instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
CMPXCHG8B <i>mem64</i>	0F C7 /1 <i>m64</i>	Compare the contents of EDX:EAX with the contents of a 64-bit memory operand. If they are equal, set the zero flag to 1 and load the contents of ECX:EBX into the 64-bit memory location. If they are not equal, load the contents of the 64-bit memory location into EDX:EAX.

Support for the CMPXCHG8B instruction depends on the processor implementation. To find out if a processor is able to execute the CMPXCHG8B instruction, use the CPUID instruction to determine whether bit 8 of CPUID standard function 1 and extended function 8000_0001h is set to 1.

Related Instructions

CMPXCHG.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The CMPXCH8B Instruction bit (bit 8) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
Undefined Opcode (#UD)	X	X	X	Destination operand is a register.

CPUID

Processor Identification

The CPUID instruction provides information about the processor implementation and its capabilities. Software can test for support of specific features by loading a function code into the EAX register and executing the CPUID instruction. Processor feature information is returned in the EAX, EBX, ECX, and EDX registers.

The architecture supports CPUID information about *standard functions* and *extended functions*. Extended functions include features such as long mode, SSE instructions, and 3DNow! instructions. Software can test for support of extended functions by first executing the CPUID instruction with the extended function 8000_0000h in the EAX register. The EAX register returns the largest extended-function input value defined for the CPUID instruction on the processor implementation. If the value in EAX is greater than 8000_0000h, extended functions are supported, although specific extended functions must be tested for individually.

Software operating at any privilege level can execute the CPUID instruction to collect this information. In 64-bit mode, this instruction works the same as legacy mode except that, if the operand size is 32 or 64 bits, 32-bit register results are zero-extended to 64 bits.

Mnemonic	Opcode	Description
CPUID	0F A2	

Some processor implementations may not support the CPUID instruction. Support is determined by testing the rFLAGS.ID bit. If software can write that bit, then the CPUID instruction is supported by the processor implementation. Otherwise, execution of CPUID results in an invalid-opcode exception.

See Section 3.3, “Processor Feature Identification”, on page 77 of Volume 2 for a detailed description of the CPUID instruction and its function codes.

Related Instructions

None.

rFLAGS Affected

None.

Exceptions

None.

CQO**Convert Quadword to Double Quadword**

This mnemonic is meaningful only in 64-bit mode.

Converts a signed quadword in RAX into a signed double quadword in RDX:RAX by copying the sign bit of RAX into all the bits of RDX. RAX is unchanged. Often used to avoid the overflow problem in signed number arithmetic.

Mnemonic	Opcode	Description
CQO	99	Sign-extends RAX into RDX.

Related Instructions

CWD, CDQ, CBW, CWDE, CDQE.

rFLAGS Affected

None.

Exceptions

None.

CWD**Convert Word to Doubleword**

Copies the sign (bit 15) of the word in the AX register into all bits of the DX register.

Often used to avoid the overflow problem in signed number arithmetic.

Mnemonic

CWD

Opcode

99

Description

Sign-extends AX into DX:AX.

Related Instructions

CDQ.

rFLAGS Affected

None.

Exceptions

None.

CWDE

Convert Word to Doubleword

Copies the sign (bit 15) of the word in the AX register into the higher 16 bits of the EAX register.

The CWDE instruction is different from the CWD (convert word to double) instruction. The CWD instruction uses the DX:AX register pair as a destination operand; whereas, the CWDE instruction uses the EAX register as a destination.

Used to convert a signed word into a signed doubleword to avoid overflow in signed number arithmetic.

In 64-bit mode, EAX is sign-extended to RAX.

Mnemonic	Opcode	Description
CWDE	98	Sign extend AX into EAX.

Related Instructions

CDQ, CWD.

rFLAGS Affected

None.

Exceptions

None.

DAA**Decimal Adjust after Addition**

Adjusts the sum of two packed BCD values to create a packed BCD result. The AL register is the implied source and destination operand. The DAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two 2-digit, packed BCD values and stores a byte result in the AL register. The DAA instruction then adjusts the contents of the AL register to contain the correct 2-digit, packed BCD result. If a decimal carry is detected, the CF and AF flags are set accordingly.

This adjustment is made by adding 6 to the lower four bits of AL if the value is greater than 9 or if AF = 1. Then it adds 6 to the upper 4 bits of AL if the upper four bits are greater than 9 or if CF = 1.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
DAA	27	Decimal adjust AL. (Invalid in 64-bit mode.)

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

DAS Decimal Adjust after Subtraction

Adjusts the result of the subtraction of two packed BCD values to create a packed BCD result. The DAS instruction is only useful when it follows a SUB instruction that subtracts (binary subtraction) one 2-digit, packed BCD value from another and stores a byte result in the AL register. The DAS instruction then adjusts the contents of the AL register to contain the correct 2-digit, packed BCD result. If a decimal borrow is detected, the CF and AF flags are set accordingly.

If the lower 4 bits of AL represent a number greater than 9 or if AF = 1, then 6 is subtracted from the lower nibble. If the upper 4 bits of AL is now greater than 9 or if CF = 1, 6 is subtracted from the upper nibble.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
DAS	2F	Decimal adjusts AL after subtraction. (Invalid in 64-bit mode.)

Related Instructions

DAA.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

DEC**Decrement by 1**

Subtracts 1 from the destination operand, while preserving the state of the CF flag. The destination operand can be a register or a memory location. This instruction allows a loop counter to be updated without disturbing the CF flag. (To perform a decrement operation that updates the CF flag, use a SUB instruction with an immediate operand of 1.)

The one-byte forms of this instruction (opcodes 48 through 4F) are used for REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.

The LOCK prefix can be used with forms of the DEC instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
DEC <i>reg/mem8</i>	FE /1	Decrement the contents of an 8-bit register or memory operand by 1.
DEC <i>reg/mem16</i>	FF /1	Decrement the contents of a 16-bit register or memory operand by 1.
DEC <i>reg/mem32</i>	FF /1	Decrement the contents of a 32-bit register or memory operand by 1.
DEC <i>reg/mem64</i>	FF /1	Decrement the contents of a 64-bit register or memory operand by 1.
DEC <i>reg16</i>	48 + <i>rw</i>	Decrement the contents of a 16-bit register operand by 1. (These opcodes are used as REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.)
DEC <i>reg32</i>	48 + <i>rd</i>	Decrement the contents of a 32-bit register operand by 1. (These opcodes are used as REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.)

Related Instructions

INC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

DIV Unsigned Divide

Divides an unsigned word, doubleword, quadword, or double quadword by an unsigned byte, word, doubleword, quadword from memory or a register. When used to divide a word, the quotient is written into AL and the remainder into AH. When used to divide a doubleword, quadword, or double quadword, the most-significant word of the dividend is in the rDX register and the least-significant word is in the rAX register. After the division, the quotient is written to rAX and the remainder into rDX.

Division by zero causes a divide-by-zero exception.

Mnemonic	Opcode	Description
<i>DIV reg/mem8</i>	F6 /6	Performs unsigned division of AX by the contents of an 8-bit register or memory operand, leaving the quotient in AL and the remainder in AH.
<i>DIV reg/mem16</i>	F7 /6	Performs unsigned division of DX:AX by the contents of a 16-bit register or memory operand, leaving the quotient in AX and the remainder in DX.
<i>DIV reg/mem32</i>	F7 /6	Performs unsigned division of EDX:EAX by the contents of a 32-bit register or memory operand, leaving the quotient in EAX and the remainder in EDX.
<i>DIV reg/mem64</i>	F7 /6	Performs unsigned division of RDX:RAX by the contents of a 64-bit register or memory operand, leaving the quotient in RAX and the remainder in RDX.

The action of this instruction depends on the operand size, as shown in the following table:

Operand Size	Dividend	Divisor	Quotient	Remainder	Quotient Range
Word/byte	AX	r/m8	AL	AH	255
Doubleword/word	DX:AX	r/m16	AX	DX	65,535
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	$2^{32} - 1$
Double quadword/quadword	RDX:RAX	r/m64	RAX	RDX	$2^{64} - 1$

Non-integral results are truncated (chopped) towards 0. The remainder is always less than the divisor in magnitude. Overflow is indicated with the #DE (divide error) exception rather than with the CF flag.

Related Instructions

IDIV, FDIV, FDIVP, FIDIV.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Divide by zero, #DE	X	X	X	Source operand (divisor) is 0.
	X	X	X	The quotient is too large for the designated register.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

ENTER**Create Procedure Stack Frame**

This instruction creates a stack frame for a procedure's local variables. The ENTER instruction takes two operands. The first operand specifies the size of the stack frame. This determines whether BP or EBP specifies the current frame pointer and whether SP or ESP specifies the stack pointer; the second operand specifies the nesting level (from 0 through 31—the depth operand is automatically masked to 5 bits).

The `enter N, 0` instruction is equivalent to the following instruction sequence:

```

push  ebp          ; save current EBP
mov    ebp, esp     ; set stack frame pointer value
sub    esp, N       ; allocate space for local variables

```

The ENTER and LEAVE instructions provide support for block structured languages. The ENTER instruction is used to set up a new stack frame for a procedure. The LEAVE instruction is then used at the before the return from the procedure to release the stack frame.

If the nesting level is 0, the processor pushes the frame pointer from the EBP register onto the stack, copies the current stack pointer from the ESP register into the EBP register, and loads the ESP register with the current stack-pointer value minus the value in the size operand.

For nesting levels of 1 or greater, the processor pushes additional frame pointers on the stack before adjusting the stack pointer. These additional frame pointers provide the called procedure with access points to other nested frames on the stack.

In 64-bit mode, the operand size defaults to 64 bits, and there is no prefix available for encoding a 32-bit operand size.

Mnemonic	Opcode	Description
ENTER <i>imm16</i> , 0	C8 <i>iw</i> 00	Create a procedure stack frame.
ENTER <i>imm16</i> , 1	C8 <i>iw</i> 01	Create a nested stack frame for a procedure.
ENTER <i>imm16</i> , <i>imm8</i>	C8 <i>iw</i> <i>ib</i>	Create a nested stack frame for a procedure.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

ENTER_START:

```

temp_ALLOC_SPACE = word-sized immediate specified in the instruction, zero-
    extended to 64 bits
temp_PARAM_COUNT = byte-sized immediate specified in the instruction, zero-
    extended to 64 bits

temp_PARAM_COUNT = temp_PARAM_COUNT AND 0x1f
                        // only keep 5 bits of param count

PUSH.v old_RBP

temp_RBP = RSP                // this value of RSP will eventually be loaded
                        // into RBP

IF (temp_PARAM_COUNT>0)      // push "temp_PARAM_COUNT" parameters to the
                        // stack
    FOR (I=1; I<temp_PARAM_COUNT; I++)
        // all but one of the parameters are copied
        // from higher up on the stack
    {
        temp_DATA = READ_MEM.v [SS:old_RBP-I*V]
        PUSH.v temp_DATA
    }
    PUSH.v temp_RBP          // the last parameter is the offset of the old
                        // value of RBP on the stack

RSP.s = RSP - temp_ALLOC_SPACE // leave "temp_ALLOC_SPACE" free bytes on
                        // the stack

temp_unused = READ_MEM.v [SS:RSP.s] // ENTER finishes with a memory check
                        // on the final stack pointer

RBP.v = temp_RBP
EXIT

```

Related Instructions

LEAVE

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	The EBP register points to a location that exceeds the limits of the current stack segment.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

IDIV Signed Divide

Divides a signed word, doubleword, quadword, or double quadword by a signed byte, word, doubleword, quadword from memory or a register. When used to divide a word, the quotient is written into AL and the remainder into AH. When used to divide a doubleword, quadword, or double quadword, the most-significant word of the dividend is in the rDX register and the least-significant word is in the rAX register. After the division, the quotient is written to rAX and the remainder into rDX.

Mnemonic	Opcode	Description
IDIV <i>reg/mem8</i>	F6 /7	Performs signed division of AX by the contents of an 8-bit register or memory operand, leaving the quotient in AL and the remainder in AH.
IDIV <i>reg/mem16</i>	F7 /7	Performs signed division of DX:AX by the contents of an 16-bit register or memory operand, leaving the quotient in AX and the remainder in DX.
IDIV <i>reg/mem32</i>	F7 /7	Performs signed division of EDX:EAX by the contents of an 32-bit register or memory operand, leaving the quotient in EAX and the remainder in EDX.
IDIV <i>reg/mem64</i>	F7 /7	Performs signed division of RDX:RAX by the contents of an 64-bit register or memory operand, leaving the quotient in RAX and the remainder in RDX.

To avoid overflow problems, it may be necessary to precede this instruction with a CBW, CWDE, or CDQE instruction to signed-extend the dividend.

Non-integral results are truncated (chopped) towards 0. The sign of the remainder is always the same as the sign of the dividend. The absolute value of the remainder is always less than the absolute value of the divisor. Overflow is indicated with the #DE (divide error) exception rather than with the OF (overflow) flag.

The action of this instruction depends on the operand size, as shown in the following table:

Operand Size	Dividend	Divisor	Quotient	Remainder	Quotient Range
Word/byte	AX	r/m8	AL	AH	-128 to +127
Doubleword/word	DX:AX	r/m16	AX	DX	-32,768 to +32,767
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	-2^{31} to $2^{32} - 1$
Double quadword/ quadword	RDX:RAX	r/m64	RAX	RDX	-2^{63} to $2^{64} - 1$

Related Instructions

IMUL, CBW, CWDE, CDQE.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								U				U	U	U	U	U
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Divide by zero, #DE	X	X	X	The divisor operand is 0.
	X	X	X	The quotient is too large for the designated register.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

IMUL Signed Multiply

Performs a signed multiplication of two operands. This instruction has three forms, depending on the number of operands.

- One-operand form. The source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, RAX register (depending on the operand size) and the product is stored in the AX, DX:AX, or EDX:EAX, RDX,RAX registers, respectively.
- Two-operand form. The destination operand (the first operand) is multiplied by the source operand (second operand). The destination is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The product is then stored in the destination operand location.
- Three-operand form. This form uses a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The product is then stored in the destination operand (a general-purpose register).

Mnemonic	Opcode	Description
IMUL <i>reg/mem8</i>	F6 /5	Multiply the contents of AL by the contents of an 8-bit memory or register operand and put the signed result in AX.
IMUL <i>reg/mem16</i>	F7 /5	Multiply the contents of AX by the contents of a 16-bit memory or register operand and put the signed result in DX: AX.
IMUL <i>reg/mem32</i>	F7 /5	Multiply the contents of EAX by the contents of a 32-bit memory or register operand and put the signed result in EDX:EAX.
IMUL <i>reg/mem64</i>	F7 /5	Multiply the contents of RAX by the contents of a 64-bit memory or register operand and put the signed 128-bit result in RDX:RAX.
IMUL <i>reg16, reg/mem16</i>	0F AF /r	Multiply the contents of a 16-bit destination register by the contents of a 16-bit register or memory operand and put the signed result in the 16-bit destination register.
IMUL <i>reg32, reg/mem32</i>	0F AF /r	Multiply the contents of a 32-bit destination register by the contents of a 32-bit register or memory operand and put the signed result in the 32-bit destination register.
IMUL <i>reg64, reg/mem64</i>	0F AF /r	Multiply the contents of a 64-bit destination register by the contents of a 64-bit register or memory operand and put the signed result in the 64-bit destination register.

Mnemonic	Opcode	Description
IMUL <i>reg16, reg/mem16, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 16-bit register or memory operand by a sign-extended immediate byte and put the signed result in the 16-bit destination register.
IMUL <i>reg32, reg/mem32, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 32-bit register or memory operand by a sign-extended immediate byte and put the signed result in the 32-bit destination register.
IMUL <i>reg64, reg/mem64, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 64-bit register or memory operand by a sign-extended immediate byte and put the signed result in the 64-bit destination register.
IMUL <i>reg16, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 16-bit register by a sign-extended immediate byte and put the signed result in the 16-bit destination register.
IMUL <i>reg32, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 32-bit register by a sign-extended immediate byte and put the signed result in the 32-bit destination register.
IMUL <i>reg64, imm8</i>	6B/r <i>ib</i>	Multiply the contents of a 64-bit register by a sign-extended immediate byte and put the signed result in the 64-bit destination register.
IMUL <i>reg16, reg/mem16, imm16</i>	69/r <i>iw</i>	Multiply the contents of a 16-bit register or memory operand by a sign-extended immediate word and put the signed result in the 16-bit destination register.
IMUL <i>reg32, reg/mem32, imm32</i>	69/r <i>id</i>	Multiply the contents of a 32-bit register or memory operand by a sign-extended immediate double and put the signed result in the 32-bit destination register.
IMUL <i>reg64, reg/mem64, imm32</i>	69/r <i>id</i>	Multiply the contents of a 64-bit register or memory operand by a sign-extended immediate double and put the signed result in the 64-bit destination register.
IMUL <i>reg16, imm16</i>	69/r <i>iw</i>	Multiply the contents of a 16-bit destination register by a 16-bit immediate value and put the signed result in the destination register.
IMUL <i>reg32, imm32</i>	69/r <i>id</i>	Multiply the contents of a 32-bit destination register by a 32-bit immediate value and put the signed result in the destination register.
IMUL <i>reg64, imm32</i>	69/r <i>id</i>	Multiply the contents of a 64-bit destination register by a 32-bit immediate value and put the signed result in the destination register.

Related Instructions

IDIV.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

IN Input from Port

Transfers a byte or word to AL or eAX from an input port specified by the second operand. The port address can be specified by an 8 bit immediate value (a value between 00h and FFh) or contained in the DX register (if greater than FFh). When the port address is kept by the DX register, the address range can be as high as FFFFh. When the port address is greater than 0FFh the address should be kept in the DX register.

If the CPL is higher than IOPL or running in virtual mode, check the IOPL bitmap.

Mnemonic	Opcode	Description
IN AL, <i>imm8</i>	E4 <i>ib</i>	Input a byte from the port at the address specified by <i>imm8</i> and put it into the AL register.
IN AX, <i>imm8</i>	E5 <i>ib</i>	Input a word from the port at the address specified by <i>imm8</i> and put it into the AX register.
IN EAX, <i>imm8</i>	E5 <i>ib</i>	Input a doubleword from the port at the address specified by <i>imm8</i> and put it into the EAX register.
IN AL, DX	EC	Input a byte from the port at the address specified by the DX register and put it into the AL register.
IN AX, DX	ED	Input a word from the port at the address specified by the DX register and put it into the AX register.
IN EAX, DX	ED	Input a doubleword from the port at the address specified by the DX register and put it into the EAX register.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	At least one of the corresponding I/O permission bits is 1 in the TSS for the accessed I/O port.

INC Increment by 1

Adds 1 to the register or memory location specified by the operand. CF is not affected, even if the operand is incremented to 0000.

The one-byte forms of this instruction (opcodes 40 through 47) are used for REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.

The LOCK prefix can be used with forms of the INC instruction that write a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Use a ADD instruction with an immediate operand of 1 to perform an increment operation that does update the CF flag.

Mnemonic	Opcode	Description
INC <i>reg/mem8</i>	FE /0	Increment the contents of an 8-bit register or memory operand by 1.
INC <i>reg/mem16</i>	FF /0	Increment the contents of a 16-bit register or memory operand by 1.
INC <i>reg/mem32</i>	FF /0	Increment the contents of a 32-bit register or memory operand by 1.
INC <i>reg/mem64</i>	FF /0	Increment the contents of a 64-bit register or memory operand by 1.
INC <i>reg16</i>	40 + <i>rw</i>	Increment the contents of a 16-bit register operand by 1. (These opcodes are used as REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.)
INC <i>reg32</i>	40 + <i>rd</i>	Increment the contents of a 32-bit register operand by 1. (These opcodes are used as REX prefixes in 64-bit mode. See “REX Prefixes” on page 14.)

Related Instructions

ADD, DEC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

INS	Input String
INSB	
INSW	
INSD	

The **INSx** instructions copy byte, word, or doubleword data from an I/O port specified in the **DX** register to an input buffer specified in the **rDI** registers.

Since segment override prefixes are ignored in 16- and 32-bit mode, the INS instruction always uses ES as the data segment. In 64-bit mode, ES/GS prefixes are ignored since, INS always uses the unsegmented memory space

After the data is transferred from the I/O port to the memory location, the eDI register is incremented or decremented automatically according to the setting of the DF flag in the rFLAGS register. If the DF flag is 0, the rDI register is incremented by 1, 2, or 4; if the DF flag is 1, the rDI register is decremented by 1, 2, or 4. The increment/decrement is equal to the number of bytes written.

The INSB instruction copies byte data from an I/O port (from 0 to 65,535) specified in the DX register into an input buffer specified in the ES:rDI registers.

The INSW instruction copies word data from an I/O port (from 0 to 65,535) specified in the DX register into an input buffer specified in the ES:rDI registers.

The INSD instruction copies doubleword data from an I/O port (from 0 to 65,535) specified in the DX register into an input buffer specified in the ES:rDI registers.

If the operand size is 64-bit, behavior is as for 32-bits.

The REP prefix can be used with the INS, INSB, INSW, and INSD instructions for block input of ECX bytes, words, or doublewords. For details about the REP prefix, see “Repeat Prefixes” on page 10.

Mnemonic	Opcode	Description
INS <i>mem8</i> , DX	6C	Input a byte from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.
INS <i>mem16</i> , DX	6D	Input a word from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.

INS <i>mem32</i> , DX	6D	Input a doubleword from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.
INSB	6C	Input a byte from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.
INSW	6D	Input a word from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.
INSD	6D	Input a doubleword from the port at the address specified by the DX register and put it into the memory location specified in ES:rDI.

The forms of the *INSx* instructions that have explicit operands are equivalent to the forms with implicit operands (in fact, they have the same opcodes). The explicit memory operand (the first operand) allows specification of the operand size. The memory operand is otherwise ignored, and instead, the implicit operand, ES:[rDI], is always used. The explicit register operand (the second operand) specifies the I/O port address and must always be DX.

The *INSx* instructions use IOPL bitmap to verify access rights if in virtual x86 mode or if IOPL is less than CPL.

Related Instructions

IN, OUT, OUTS, OUTSB, OUTSD, OUTSW.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X	X	Bad memory address

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	IOPL bits are set
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

INT Interrupt to Vector

Transfers execution to one of the 256 interrupt service routines. The address of the service routine is specified via the interrupt vector number, which can be between 00h and FFh.

Mnemonic	Opcode	Description
INT <i>imm8</i>	CD <i>ib</i>	Call interrupt service routine specified by interrupt vector <i>imm8</i> .

For detailed descriptions of the steps performed by INT n instructions, see the following:

- *Legacy-Mode Interrupts*: Section 8.7, “Legacy-Mode Interrupt Control Transfers”, on page 285 of Volume 2.
- *Long-Mode Interrupts*: Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.

See also the descriptions of the INT 3 instruction on page 1054 and the INTO instruction on page 147.

Note that if the interrupt causes a task switch all EFLAGS may be modified.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

INT_N_START:

```
IF (REAL_MODE)
    INT_N_REAL
ELSIF (PROTECTED_MODE)
    INT_N_PROTECTED
ELSE // (VIRTUAL_MODE)
    INT_N_VIRTUAL
```

```
INT_N_REAL:
    temp_int_n_vector = byte-sized interrupt vector specified in the instruction,
    zero-extended to 64 bits

    temp_RIP = READ_MEM.w [idt:temp_int_n_vector*4]
                // read target CS:RIP from the real-mode idt
    temp_CS = READ_MEM.w [idt:temp_int_n_vector*4+2]
    PUSH.w old_RFLAGS
    PUSH.w old_CS
    PUSH.w next_RIP
```

```

IF (temp_RIP>CS.limit)
    EXCEPTION [#GP]
CS.sel = temp_CS
CS.base = temp_CS SHL 4
RFLAGS.AC,TF,IF,RF cleared
RIP = temp_RIP
EXIT

```

INT_N_PROTECTED:

```

temp_int_n_vector = byte-sized interrupt vector specified in the instruction,
zero-extended to 64 bits
temp_idt_desc = READ_IDT (temp_int_n_vector)
IF (temp_idt_desc.attr.type = 'taskgate')
    TASK_SWITCH // using tss selector in the task gate as the target tss
IF (LONG_MODE) // the size of the gate controls the size of the stack pushes
    V=8-byte // long mode only uses 64-bit gates
ELIF ((temp_idt_desc.attr.type = 'intgate32')
    || (temp_idt_desc.attr.type = 'trapgate32'))
    V=4-byte // legacy mode, using a 32-bit gate
ELSE // gate is intgate16 or trapgate16
    V=2-byte // legacy mode, using a 16-bit gate

temp_RIP = temp_idt_desc.offset
IF (LONG_MODE)
    // in long mode, we need to read the 2nd half of a
    // 16-byte interrupt-gate from the gdt/ldt, to get the
    // upper 32 bits of the target RIP
{
    temp_upper = READ_MEM.q [idt:temp_int_n_vector*16+8]
    IF (temp_upper's extended attribute bits != 0)
        EXCEPTION [#GP(0)] // the extended attribute bits must be all zero
    temp_RIP = temp_RIP + (temp_upper SHL 32) // concatenate both halves of RIP
}

CS = READ_DESCRIPTOR (temp_idt_desc.segment, intcs_chk)

temp_CPL = CS.sel.rpl

IF (CPL=temp_CPL) // no privilege-level change
    IF (LONG_MODE)
        IF (temp_idt_desc.ist!=0)
            // in long mode, if the idt gate specifies an ist pointer,
            // a stack-switch is always done
            RSP = READ_MEM.q [tss:ist_index*8+28]

            RSP = RSP AND 0xFFFFFFFFFFFFFFF0
            // in long mode, interrupts/exceptions align rsp to a
            // 16-byte boundary

```

```

        PUSH.q old_SS      // in long mode, SS:RSP is always pushed to the stack
        PUSH.q old_RSP

    PUSH.v old_RFLAGS
    PUSH.v old_CS
    PUSH.v next_RIP

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
        EXCEPTION [#GP(0)]

    RFLAGS.VM,NT,TF,RF cleared
    RFLAGS.IF cleared if interrupt gate

    RIP = temp_RIP
    EXIT

ELSE // (CPL > temp_CPL), changing privilege level
{
    CPL = temp_CPL

    temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER
                          (CPL, temp_idt_desc.ist)

    IF (LONG_MODE)
        temp_RSP = temp_RSP AND 0xFFFFFFFFFFFFFFF0
                // in long mode, interrupts/exceptions align rsp
                // to a 16-byte boundary

    RSP.q = temp_RSP
    SS = temp_SS_desc

    PUSH.v old_SS      // #SS on the following pushes uses SS.sel as error code
    PUSH.v old_RSP
    PUSH.v old_RFLAGS
    PUSH.v old_CS
    PUSH.v next_RIP

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
        EXCEPTION [#GP(0)]

    RFLAGS.VM,NT,TF,RF cleared
    RFLAGS.IF cleared if interrupt gate
    RIP = temp_RIP
    EXIT
}

```

INT_N_VIRTUAL:

```

temp_int_n_vector = byte-sized interrupt vector specified in the instruction,
    zero-extended to 64 bits

IF (CR4.VME=0)                // vme isn't enabled
    IF (RFLAGS.IOPL=3)
        INT_N_VIRTUAL_TO_PROTECTED
    ELSE
        EXCEPTION [#GP(0)]

temp_IRB_BASE = READ_MEM.w [tss:102]
    // check the vme Int-n Redirection Bitmap (IRB), to see
    // if we should redirect this interrupt to a virtual-mode
    // handler
temp_VME_REDIRECTION_BIT = READ_BIT_ARRAY ([tss:temp_IRB_BASE-32],
    temp_int_n_vector)

IF (temp_VME_REDIRECTION_BIT=1)
    // the virtual-mode int-n bitmap bit is set, so don't
    // redirect this interrupt
    IF (RFLAGS.IOPL=3)
        INT_N_VIRTUAL_TO_PROTECTED
    ELSE
        EXCEPTION [#GP(0)]
ELSE
    // redirect interrupt through virtual-mode idt
{
    temp_RIP = READ_MEM.w [0:temp_int_n_vector*4]
    // read target CS:RIP from the virtual-mode idt at
    // linear address 0
    temp_CS = READ_MEM.w [0:temp_int_n_vector*4+2]

    IF (RFLAGS.IOPL < 3)
        old_RFLAGS = old_RFLAGS with VIF bit shifted into IF bit, and IOPL = 3

    PUSH.w old_RFLAGS
    PUSH.w old_CS
    PUSH.w next_RIP

    CS.sel = temp_CS
    CS.base = temp_CS SHL 4

    RFLAGS.TF,IF,RF cleared
    RIP = temp_RIP        // RFLAGS.IF cleared if IOPL = 3
    // RFLAGS.VIF cleared if IOPL < 3
    EXIT
}

```

INT_N_VIRTUAL_TO_PROTECTED:

```

temp_idt_desc = READ_IDT (temp_int_n_vector)
IF (temp_idt_desc.attr.type = 'taskgate')
    TASK_SWITCH // using tss selector in the task gate as the target tss

IF ((temp_idt_desc.attr.type = 'intgate32')
    || (temp_idt_desc.attr.type = 'trapgate32'))
    // the size of the gate controls the size of the stack pushes
    V=4-byte // legacy mode, using a 32-bit gate
ELSE // gate is intgate16 or trapgate16
    V=2-byte // legacy mode, using a 16-bit gate

temp_RIP = temp_idt_desc.offset
CS = READ_DESCRIPTOR (temp_idt_desc.segment, intcs_chk)

IF (CS.sel.rpl!=0) // handler must run at cpl 0
    EXCEPTION [#GP(0)]

CPL = 0

temp_ist = 0 // legacy mode doesn't use ist pointers
temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER (CPL, temp_ist)

RSP.q = temp_RSP
SS = temp_SS_desc

PUSH.v old_GS // #SS on the following pushes use SS.sel as error code
PUSH.v old_FS
PUSH.v old_DS
PUSH.v old_ES
PUSH.v old_SS
PUSH.v old_RSP
PUSH.v old_RFLAGS // pushed with RF clear
PUSH.v old_CS
PUSH.v next_RIP

IF (temp_RIP > CS.limit)
    EXCEPTION [#GP(0)]

DS = NULL // can't use virtual-mode selectors in protected mode
ES = NULL // can't use virtual-mode selectors in protected mode
FS = NULL // can't use virtual-mode selectors in protected mode
GS = NULL // can't use virtual-mode selectors in protected mode

RFLAGS.VM,NT,TF,RF cleared
RFLAGS.IF cleared if interrupt gate

RIP = temp_RIP
EXIT

```


Related Instructions

INT 3, INTO, BOUND.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
					0	0				1/0	1/0					
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid TSS, #TS (selector)		X	X	The TSS's stack-segment selector is null.
		X	X	The new stack-segment selector in the TSS contains an RPL that is not equal to the DPL of the code-segment descriptor that is being accessed by the interrupt or trap gate by the interrupt or trap gate.
		X	X	The target stack-segment descriptor contains a DPL that is not equal to that of the code-segment descriptor being accessed by the interrupt or trap gate.
		X	X	The TSS's stack-segment is not a writable data segment.
		X	X	The index in the stack-segment selector is beyond the descriptor table's limits.
Segment not present, #NP (selector)		X	X	One of the following is not present: code segment, TSS, trap gate, task gate, or interrupt gate.
Stack, #SS	X		X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)		X		The stack segment limit was exceeded when a stack switch occurred.
			X	The target stack segment is not present.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The instruction pointer in the in the task gate, interrupt gate, trap gate, or IDT is outside the limits of the code segment.
	X			During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
	X			The exception vector is beyond the limits of the IDT.
		X		If the IOPL is less than 3 and the DPL of the interrupt-, trap-, or task-gate descriptor is not equal to 3.
General protection, #GP (selector)		X	X	The segment selector is null in the trap, interrupt, or task gate.
		X	X	The selector index for the TSS, gate, or code segment, is beyond the limits of the descriptor table.
		X	X	The exception vector is beyond the limits of the IDT.
		X	X	The descriptor in the IDT is not for an interrupt, trap, or task gate.
		X	X	An interrupt is generated by the INTn instruction and the DPL of an interrupt, trap, or task gate is less than the CPL. (For protected mode, also INT 3 or INTO instruction)
		X	X	The segment selector in a trap gate or interrupt does not point to a code-segment descriptor.
		X	X	The TI bit in the TSS's segment selector indicates a local table.
			X	The segment descriptor for the TSS indicates that the TSS is not available or busy. (Protected mode only)
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

INTO**Interrupt to Overflow Vector**

Checks the overflow flag (OF) in the rFLAGS register and calls the overflow exception (#OF) handler if the OF flag is set to 1. The INTO instruction is used to detect overflow in signed number addition. In signed addition, OF is set to 1 in two cases:

- Whenever there is a carry from bit 6 to bit 7 in 8-bit operations and no carry from bit 7 out (or in 16-bit operations when there is carry from bit 14 to bit 15 and CF = 0).
- When there is carry from bit 7 out and no carry from bit 6 to bit 7 (or in the case of 16-bit operation when there is a carry from bit 15 out and no carry from bit 14 to bit 15).

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
INTO	CE	Call overflow exception if the overflow flag is set. (Invalid in 64-bit mode.)

For detailed descriptions of the steps performed by INT instructions, see the following:

- *Legacy-Mode Interrupts*: Section 8.7, “Legacy-Mode Interrupt Control Transfers”, on page 285 of Volume 2.
- *Long-Mode Interrupts*: Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.

Related Instructions

INT, INT 3, BOUND.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
										1/0	1/0					
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Breakpoint, #BP		X		INT 3 instruction is executed.
Overflow, #OF		X		The INTO instruction is executed.
Invalid opcode, #UD			X	Software is executing in 64-bit mode.
Invalid TSS, #TS (selector)		X	X	The TSS's stack-segment selector is null.
		X	X	The new stack-segment selector in the TSS contains an RPL that is not equal to the DPL of the code-segment descriptor that is being accessed by the interrupt or trap gate.
		X	X	The target stack-segment descriptor contains a DPL that is not equal to that of the code-segment descriptor being accessed by the interrupt or trap gate.
		X	X	The TSS's stack-segment is not a writable data segment.
		X	X	The index in the stack-segment selector is beyond the descriptor table's limits.
Segment not present, #NP (selector)		X	X	One of the following is not present: code segment, TSS, trap gate, task gate, or interrupt gate.
Stack, #SS	X		X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)		X		The stack segment limit was exceeded when a stack switch occurred.
			X	The target stack segment is not present.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	<p>The instruction pointer in the in the task gate, interrupt gate, trap gate, or IDT is outside the limits of the code segment.</p> <p>The INT4 vector is beyond the limits of the IDT.</p> <p>If the IOPL is less than 3 and the DPL of the interrupt-, trap-, or task-gate descriptor is not equal to 3.</p>
General protection, #GP (selector)		X	X	The segment selector is null in the trap, interrupt, or task gate.
		X	X	The selector index for the TSS, gate, or code segment, is beyond the limits of the descriptor table.
		X	X	The exception vector is beyond the limits of the IDT.
		X	X	The descriptor in the IDT is not for an interrupt, trap, or task gate.
		X	X	The DPL of an interrupt, trap, or task gate is less than the CPL.
		X	X	The segment selector in a trap gate or interrupt does not point to a code-segment descriptor.
		X	X	The TI bit in the TSS's segment selector indicates a local table.
		X	X	The segment descriptor for the TSS indicates that the TSS is not available or busy.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Jcc Jump on Condition

Jumps to a relative target address, depending upon the status of the arithmetic flags in the rFLAGS register. Unlike the unconditional jump (JMP), conditional jump instructions have only two forms—*short and near conditional jumps*. Different opcodes correspond to different forms of one instruction. For example, the JO instruction (jump if overflow) has opcode 0Fh 80h for its near form and 70h for its short form, but the mnemonic is the same for both forms.

Mnemonics are provided to deal with the programming semantics of both signed and unsigned numbers. Instructions tagged A (above) and B (below) are intended to clarify unsigned integer code; those tagged G (greater) and L (less) are intended for use in signed integer manipulations.

In 64-bit mode, the operand size defaults to 64 bits. The resulting RIP = starting RIP + 8-bit or 32-bit displacement sign-extended to 64 bits. (The 32-bit displacement cannot be encoded because the operand size defaults to 64 bits.)

Mnemonic	Opcode	Description
JA <i>rel8off</i> JNB <i>rel8off</i>	77 <i>cb</i>	Jump short if the carry flag and the zero flag are both zero.
JA <i>rel16off</i> JNB <i>rel16off</i>	0F 87 <i>cw</i>	Jump near if the carry flag and the zero flag are both zero.
JA <i>rel32off</i> JNB <i>rel32off</i>	0F 87 <i>cd</i>	Jump near if the carry flag and the zero flag are both zero.
JA <i>rel8off</i> JNB <i>rel8off</i> JNC <i>rel8off</i>	73 <i>cb</i>	Jump short if the carry flag is zero.
JA <i>rel16off</i> JNB <i>rel16off</i> JNC <i>rel16off</i>	0F 83 <i>cw</i>	Jump near if the carry flag is zero.
JA <i>rel32off</i> JNB <i>rel32off</i> JNC <i>rel32off</i>	0F 83 <i>cd</i>	Jump near if the carry flag is zero.
JBE <i>rel8off</i>	76 <i>cb</i>	Jump short if the carry flag is 1 or the zero flag is 1.
JBE <i>rel16off</i>	0F 86 <i>cw</i>	Jump near if the carry flag is 1 or the zero flag is 1.
JBE <i>rel32off</i>	0F 86 <i>cd</i>	Jump near if the carry flag is 1 or the zero flag is 1.
JC <i>rel8off</i> JB <i>rel8off</i> JNAE <i>rel8off</i>	72 <i>cb</i>	Jump short if the carry flag is 1.

JC <i>rel16off</i> JB <i>rel16off</i> JNAE <i>rel16off</i>	0F 82 <i>cw</i>	Jump near if the carry flag is 1.
JC <i>rel32off</i> JB <i>rel32off</i>	0F 82 <i>cd</i>	Jump near if the carry flag is 1.
JE <i>rel8off</i> JZ <i>rel8off</i>	74 <i>cb</i>	Jump short if the zero flag is 1.
JE <i>rel16off</i> JZ <i>rel16off</i>	0F 84 <i>cw</i>	Jump near if the zero flag is 1.
JE <i>rel32off</i> JZ <i>rel32off</i>	0F 84 <i>cd</i>	Jump near if the zero flag is 1.
JG <i>rel8off</i> JNLE <i>rel8off</i>	7F <i>cb</i>	Jump short if the zero flag is zero and the sign flag is equal to the overflow flag.
JG <i>rel16off</i> JNLE <i>rel16off</i>	0F 8F <i>cw</i>	Jump near if the zero flag is zero and the sign flag is equal to the overflow flag.
JG <i>rel32off</i> JNLE <i>rel32off</i>	0F 8F <i>cd</i>	Jump near if the zero flag is zero and the sign flag is equal to the overflow flag.
JGE <i>rel8off</i> JNL <i>rel8off</i>	7D <i>cb</i>	Jump short if the sign flag and the overflow flag have the same value.
JGE <i>rel16off</i> JNL <i>rel16off</i>	0F 8D <i>cw</i>	Jump near if the sign flag and the overflow flag have the same value.
JGE <i>rel32off</i> JNL <i>rel32off</i>	0F 8D <i>cd</i>	Jump near if the sign flag and the overflow flag have the same value.
JL <i>rel8off</i> JNGE <i>rel8off</i>	7C <i>cb</i>	Jump short if the sign flag is not equal to the overflow flag.
JL <i>rel16off</i> JNGE <i>rel16off</i>	0F 8C <i>cw</i>	Jump near if the sign flag is not equal to the overflow flag.
JL <i>rel32off</i> JNGE <i>rel32off</i>	0F 8C <i>cd</i>	Jump near if the sign flag is not equal to the overflow flag.
JLE <i>rel8off</i> JNG <i>rel8off</i>	7E <i>cb</i>	Jump short if the zero flag is 1 and sign flag is not equal to the overflow flag.
JLE <i>rel16off</i> JNG <i>rel16off</i>	0F 8E <i>cw</i>	Jump near if the zero flag is 1 and the sign flag is not equal to the overflow flag.
JLE <i>rel32off</i> JNG <i>rel32off</i>	0F 8E <i>cd</i>	Jump near if the zero flag is 1 and the sign flag is not equal to the overflow flag.
JNE <i>rel8off</i> JNZ <i>rel8off</i>	75 <i>cb</i>	Jump short if the zero flag is zero.
JNE <i>rel16off</i> JNZ <i>rel16off</i>	0F 85 <i>cw</i>	Jump near if the zero flag is zero.

JNE <i>rel32off</i>	0F 85 <i>cd</i>	Jump near if the zero flag is zero.
JNZ <i>rel32off</i>		
JNO <i>rel8off</i>	71 <i>cb</i>	Jump short if the overflow flag is zero.
JNO <i>rel16off</i>	0F 81 <i>cw</i>	Jump near if the overflow flag is zero.
JNO <i>rel32off</i>	0F 81 <i>cd</i>	Jump near if the overflow flag is zero.
JNP <i>rel8off</i>	7B <i>cb</i>	Jump short if the parity flag is zero.
JPO <i>rel8off</i>		
JNP <i>rel16off</i>	0F 8B <i>cw</i>	Jump near if the parity flag is zero.
JPO <i>rel16off</i>		
JNP <i>rel32off</i>	0F 8B <i>cd</i>	Jump near if the parity flag is zero.
JPO <i>rel32off</i>		
JS <i>rel8off</i>	78 <i>cb</i>	Jump short if the sign flag is 1.
JS <i>rel16off</i>	0F 88 <i>cw</i>	Jump near if the sign flag is 1.
JC <i>rel32off</i>	0F 88 <i>cd</i>	Jump near if the sign flag is 1.

These instructions do not support far jumps (jumps to other code segments). To create a far-conditional-jump code sequence corresponding to a high-level language statement like:

```
IF A = B THEN GOTO FarLabel
```

where *FarLabel* is located in another code segment, use the opposite condition in a conditional short jump before the unconditional far jump. Such a code sequence might look like:

```
cmp    A,B           ; compare operands
jne    NextInstr     ; continue program if not equal
jmp    far ptr WhenNE ; far jump if operands are equal
```

NextInstr;; continue program

For details about control-flow instructions, see Section 3.7, “Control Transfers”, on page 95 of Volume 1, and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.

Related Instructions

JCXZ, JECXZ, JMP, JRCXZ

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X		The offset being jumped to is beyond the limits of the CS segment or is outside of the effective address space of 0 to FFFFh. This condition can occur if a 32-bit address-size-override prefix is used.
			X	The offset being jumped to is beyond the limits of the CS segment.

JCXZ Jump if CX Zero

JECXZ

JRCXZ

Jumps to an 8-bit displacement if the count register (rCX) is zero. The target address cannot be more than –128 to +127 bytes away.

In 64-bit mode, the operand size defaults to 64 bits. The resulting RIP = starting RIP + 8-bit displacement sign-extended to 64 bits. The resulting RIP is then masked to 16 or 32 bits, if the operand size is 16 or 32 bits.

Mnemonic	Opcode	Description
JCXZ <i>rel8off</i>	E3 <i>cb</i>	Jump short if the count register (CX) is zero.
JECXZ <i>rel8off</i>	E3 <i>cb</i>	Jump short if the extended count register (ECX) is zero.
JRCXZ <i>rel8off</i>	E3 <i>cb</i>	Jump short if the extended count register (RCX) is zero.

For details about control-flow instructions, see Section 3.7, “Control Transfers”, on page 95 of Volume 1, and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.

Related Instructions

Jcc, JMP

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	The offset being jumped to is beyond the limits of the CS segment.

address.

- *Relative Near Jump*—The target address must be within a 16-bit or 32-bit signed displacement of the rIP of the instruction after the JMP. This is a three-byte instruction. The first byte is opcode E9h and the next two bytes are the signed-number displacement. The displacement is added to the rIP of the instruction following the JMP to get the target address.
- *Register-Indirect Near Jump*—The target address is contained in a register. Any non-segment register can be used to specify the target address.
- *Memory-Indirect Near Jump*—The target address is in a memory location whose address is pointed to by a register.
- *Direct Far Jump*—The target address is outside the current code segment. Therefore, software must specify both the code segment and the offset of the target address. This is a five-byte instruction: the opcode EAh and 4 bytes for the segment and offset of the target address.
- *Memory-Indirect Far Jump*—The target address (both CS:rIP) is in a memory location whose address is pointed to by a register.

For near jumps in 64-bit mode, the operand size defaults to 64 bits. For the EB opcode, the resulting RIP = starting RIP + 8-bit displacement sign-extended to 64 bits. For the E9 opcode, the resulting RIP = starting RIP + 32-bit displacement sign-extended to 64 bits (the 16-bit displacement cannot be encoded because the operand size defaults to 64 bits). For the FF /4 opcode, the resulting RIP = the 64-bit offset from the register or memory.

For far jumps in 64-bit mode, the EA opcode is invalid, and the FF /5 opcode does the following:

- For 32-bit operand size, if the selector points to a gate, then RIP = zero-extended 32-bit offset from the gate, else RIP = zero-extended 32-bit offset from the far pointer referenced in instruction.
- For 64-bit operand size, if the selector points to a gate, then RIP = 64-bit offset from the gate, else RIP = zero-extended 32-bit offset from the far pointer referenced in instruction.

For details about control-flow instructions, see Section 3.7, “Control Transfers”, on page 95 of Volume 1, and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.

Action

```
// Far jumps (JMPF)
// See Section 2.5.3, “Pseudocode Definitions”, on page 46.
```

```
JMPF_START:
```

```

IF (REAL_MODE)
    JMPF_REAL_OR_VIRTUAL
ELSIF (PROTECTED_MODE)
    JMPF_PROTECTED
ELSE // (VIRTUAL_MODE)
    JMPF_REAL_OR_VIRTUAL

```

JMPF_REAL_OR_VIRTUAL:

```

IF (OPCODE = jmpf [mem])
{
    temp_RIP = READ_MEM.z [mem]
    temp_CS  = READ_MEM.w [mem+Z]
}
ELSE // (OPCODE = jmpf direct)
{
    temp_RIP = z-sized offset specified in the instruction,
               zero-extended to 64 bits
    temp_CS  = selector specified in the instruction
}

IF (temp_RIP > CS.limit)
    EXCEPTION [#GP(0)]

CS.sel = temp_CS
CS.base = temp_CS SHL 4
RIP = temp_RIP
EXIT

```

JMPF_PROTECTED:

```

IF (OPCODE = jmpf [mem])
{
    temp_offset = READ_MEM.z [mem]
    temp_sel    = READ_MEM.w [mem+Z]
}
ELSE // (OPCODE = jmpf direct)
{
    IF (64BIT_MODE)
        EXCEPTION [#UD] // 'jmpf direct' is illegal in 64-bit mode
    temp_offset = z-sized offset specified in the instruction,
                   zero-extended to 64 bits
    temp_sel    = selector specified in the instruction
}
temp_desc = READ_DESCRIPTOR (temp_sel, cs_chk)
           // read descriptor, perform protection and type checks
IF (temp_desc.attr.type = 'available_tss')
    TASK_SWITCH // using temp_sel as the target tss selector
ELSIF (temp_desc.attr.type = 'taskgate')

```

```

        TASK_SWITCH        // using the tss selector in the task gate as the
                           // target tss
    ELSIF (temp_desc.attr.type = 'code')
        // if the selector refers to a code descriptor, then
        // the offset we read is the target RIP
    {
        temp_RIP = temp_offset
        CS = temp_desc
        IF ((!64BIT_MODE) && (temp_RIP > CS.limit))
            // temp_RIP can't be non-canonical because
            // it's a 16- or 32-bit number, zero-extended to 64 bits
        {
            EXCEPTION [#GP(0)]
        }
        RIP = temp_RIP
        EXIT
    }
    ELSE
    {
        // (temp_desc.attr.type = 'callgate')
        // if the selector refers to a call gate, then
        // the target CS and RIP both come from the call gate
        temp_RIP = temp_desc.offset

        IF (LONG_MODE)
        {
            // in long mode, we need to read the 2nd half of a 16-byte call-gate
            // from the gdt/ldt to get the upper 32 bits of the target RIP
            temp_upper = READ_MEM.q [temp_sel+8]
            IF (temp_upper [44:40] != 0)
                EXCEPTION [#GP(0)] // make sure the extended attribute bits
                                   // are all zero
            temp_RIP = temp_RIP + (temp_upper SHL 32)
            // concatenate both halves of RIP
        }
        CS = READ_DESCRIPTOR (temp_desc.segment, clg_chk)
            // set up new CS base, attr, limits
        IF ((64BIT_MODE) && (temp_RIP is non-canonical)
            || (!64BIT_MODE) && (temp_RIP > CS.limit))
            EXCEPTION [#GP(0)]
        RIP = temp_RIP
        EXIT
    }

```

Related Instructions

Jcc, JCXZ, JECXZ, JRCXZ.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	The EA opcode of this instruction is not recognized in 64-bit mode.
Segment not present, #NP (selector)			X	The accessed code segment, call gate, task gate, or TSS is not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP		X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The offset in the TSS, call gate, or target operand exceeds the limits of the code segment.
			X	There is a null segment selector in the destination operand, TSS, call gate, or task gate.
			X	There is a null segment selector in a data-segment register that is accessing memory.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP (selector)			X	The segment selector index is outside the descriptor table limits.
			X	The segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.
			X	The DPL for a nonconforming-code segment is not equal to the CPL.
			X	When not using a call gate, the RPL for the segment's segment selector is greater than the CPL.
			X	A conforming-code segment's DPL is greater than the CPL.
			X	The DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL, or than the RPL of the call-gate, task-gate, or TSS segment selector.
			X	The segment descriptor for selector in a call gate does not indicate it is a code segment.
			X	The segment descriptor for the segment selector in the task gate does not indicate available TSS.
			X	The TI bit in the TSS's segment selector indicates a local table.
			X	The segment descriptor for the TSS indicates that the TSS is not available or busy.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. This exception only occurs when fetching target from memory. (In Protected Mode, CPL = 3.)

LAHF Load Status Flags into AH Register

Loads the lower 8 bits of the rFLAGS register, including the trap flag (TR), sign flag (SF), zero flag (ZF), auxiliary carry flag (AF), parity flag (PF), and carry flag (CF), into the AH register.

Reserved bits 1, 3, and 5 of the EFLAGS register are set in the AH register as 1, 0, and 0, respectively.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
LAHF	9F	Load the SF, ZF, AF, PF, and CF flags into the AH register. (Invalid in 64-bit mode.)

Related Instructions

SAHF.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

LDS Load DS Far Pointer

Loads the contents of two successive memory locations into the DS segment register and a general-purpose register. This is useful for accessing a new data segment and its offset.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

If this instruction is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register. Also in protected mode, a null selector (values 0000 through 0003) can be loaded into the DS register without causing a protection exception. Any subsequent reference to the segment whose corresponding register is loaded with a null selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.

Mnemonic	Opcode	Description
LDS <i>reg16, mem16:16</i>	C5/r	Load DS:reg16 with a specified far pointer from memory. (Invalid in 64-bit mode.)
LDS <i>reg32, mem16:32</i>	C5/r	Load DS:reg16 with a specified far pointer from memory. (Invalid in 64-bit mode.)

Related Instructions

LEA, LES, LFS, LGS, LSS, STOS, STOSB, STOSW, STOSD.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.
			X	Software is executing in 64-bit mode.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and the segment is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS (selector)			X	The SS register is being loaded and the segment is marked not present.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LEA Load Effective Address

Computes the effective address of the source operand and stores it in the destination operand.

The destination operand is a general-purpose register, and the source operand is a memory location (with a 16-bit, 32-bit, or 64-bit address-size attribute) that contains the effective address. If the source operand is not a memory location (i.e., it is a register), an undefined-opcode exception occurs.

Mnemonic	Opcode	Description
<code>LEA reg16, mem16</code>	8D/r	Store effective <i>mem16:32</i> address in a 16-bit destination register.
<code>LEA reg32, mem32</code>	8D/r	Store effective <i>mem16:32</i> address in a 32-bit destination register.
<code>LEA reg64, mem64</code>	8D/r	Store effective <i>mem16:64</i> address in a 64-bit destination register.

The result of performing the LEA instruction depends on the address size of the source operand and the operand size of the destination operand in the following way:

- If the address size of source and the operand size of the destination are the same, no size conversion is performed on the destination operand.
- If the address size of the source is longer than the operand size of the destination, the effective address stored in the destination is truncated to the operand size of the destination.
- If the address size of the source is shorter than the operand size of the destination, the effective address stored in the destination is zero-extended to the size of the destination.

LEA is related to MOV, which copies data from a memory location to a register, but LEA takes the address of the source operand, whereas MOV takes the contents of the memory location specified by the source operand. In the simplest cases, LEA can be replaced with MOV. For example:

```
lea eax, [ebx]
```

has the same effect as:

```
mov eax, ebx
```

However, LEA allows software to use any valid addressing mode for the source operand. For example:

```
leal eax, [ebx+edi]
```

loads the sum of EBX and EDI registers into the EAX register. This could not be accomplished by a single MOV instruction.

LEA has a limited capability to perform multiplication of operands in general-purpose registers using scaled-index addressing. For example:

```
leal eax, [ebx+ebx*8]
```

loads the value of the EBX register, multiplied by 9, into the EAX register. Possible values of multipliers are 2, 4, 8, 3, 5, and 9.

LEA is widely used in string-processing and array-processing for initializing an index register (rSI or rDI) before performing string instructions such as MOVSx. LEA is also used to initialize the rBX register before performing the XLAT instruction in programs that perform character translations. In data structures, LEA can calculate addresses of operands stored in memory, and in particular, addresses of array or string elements.

Related Instructions

LDS, LES, LFS, LGS, LSS, STOS, STOSB, STOSW, STOSD.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.

LEAVE**Delete Procedure Stack Frame**

Releases the stack frame set up by an earlier ENTER instruction. The LEAVE instruction copies the frame pointer (in the EBP register) into the stack pointer register (ESP), which releases the stack space allocated to the stack frame. The old frame pointer (the frame pointer for the calling procedure that was saved by the ENTER instruction) is then popped from the stack into the EBP register, restoring the calling procedure's stack frame.

A RET instruction is commonly executed following a LEAVE instruction to return program control to the calling procedure.

In 64-bit mode, the operand size defaults to 64 bits, and there is no prefix available for encoding a 32-bit operand size.

Mnemonic	Opcode	Description
LEAVE	C9	Set the stack pointer register SP to the value in the BP register and pop BP.
LEAVE	C9	Set the stack pointer register ESP to the value in the EBP register and pop EBP. (No prefix for encoding this in 64-bit mode.)
LEAVE	C9	Set the stack pointer register RSP to the value in the RBP register and pop RBP.

Related Instructions

ENTER.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS			X	The EBP register points to a location that exceeds the limits of the current stack segment.
General protection, segment overrun, #GP	X	X		The EBP register points to a location outside of the effective address space from 0 to FFFFh.

LES Load ES Far Pointer

Loads the contents of two successive memory locations into the ES segment register and a general-purpose register. Useful for accessing a new extra segment and its offset. This instruction is similar to LDS except that the ES and its offset are being loaded.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

If this instruction is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register. Also in protected mode, a null selector (values 0000 through 0003) can be loaded into the ES register without causing a protection exception. Any subsequent reference to the segment whose corresponding register is loaded with a null selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.

Mnemonic	Opcode	Description
LES <i>reg16, mem16:16</i>	C4/ <i>r</i>	Load ES:reg16 with a specified far pointer from memory. (Invalid in 64-bit mode.)
LES <i>reg32, mem16:32</i>	C4/ <i>r</i>	Load ES:reg16 with a specified far pointer from memory. (Invalid in 64-bit mode.)

Related Instructions

LDS, LFS, LGS, LSS.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.
			X	Software is executing in 64-bit mode.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and the segment is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)			X	The SS register is being loaded and the segment is marked not present.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LFENCE**Load Fence**

The LFENCE instruction acts as a barrier to force strong memory ordering (serialization) between load instructions preceding the LFENCE and load instructions that follow the LFENCE. In a weakly-ordered memory system, hardware is allowed to reorder reads and writes between the processor and memory. The LFENCE instruction guarantees that all prior loads complete before subsequent loads are executed.

LFENCE is weakly-ordered with respect to store instructions, data and instruction prefetches, and the SFENCE instruction. Speculative loads initiated by the processor, or specified explicitly using cache-prefetch instructions, can be reordered around an LFENCE.

In addition to load instructions, LFENCE is strongly ordered with respect to other LFENCE instructions, MFENCE instructions, and serializing instructions.

Support for the LFENCE instruction is indicated when the SSE2 bit (bit 26) is set to 1 in EDX after executing CPUID with extended function 8000_0001h.

Mnemonic	Opcode	Description
LFENCE	0F AE E8	Force strong ordering of (serialize) load operations.

Related Instructions

MFENCE, SFENCE

rFLAGS Affected

None.

Exceptions

None.

LFS Load FS Far Pointer

Loads the contents of two successive memory locations into the FS segment register and a general-purpose register. Useful for accessing a new extra segment and its offset. This instruction is similar to LDS except that the FS and its offset are being loaded.

In 64-bit mode, for both 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits.

If this instruction is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register. Also in protected mode, a null selector (values 0000 through 0003) can be loaded into the FS register without causing a protection exception. Any subsequent reference to the segment whose corresponding register is loaded with a null selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.

Mnemonic	Opcode	Description
LFS <i>reg16, mem16:16</i>	0F B4/ <i>r</i>	Load FS:reg16 with a specified far pointer from memory.
LFS <i>reg32, mem16:32</i>	0F B4/ <i>r</i>	Load FS:reg32 with a specified far pointer from memory.
LFS <i>reg64, mem16:32</i>	0F B4/ <i>r</i>	Load FS:reg32 with a specified far pointer from memory.

Related Instructions

LDS, LES, LGS, LSS.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and the segment is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)			X	The SS register is being loaded and the segment is marked not present.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LGS Load GS Far Pointer

Loads the contents of two successive memory locations into the GS segment register and a general-purpose register. Useful for accessing a new extra segment and its offset. This instruction is similar to LDS except that the GS and its offset are being loaded.

In 64-bit mode, for both 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits.

If this instruction is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register. Also in protected mode, a null selector (values 0000 through 0003) can be loaded into the GS register without causing a protection exception. Any subsequent reference to the segment whose register is loaded with a null selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.

Mnemonic	Opcode	Description
<i>LGS reg16, mem16:16</i>	0F B5/r	Load GS:reg16 with a specified far pointer from memory.
<i>LGS reg32, mem16:32</i>	0F B5/r	Load GS:reg32 with a specified far pointer from memory.
<i>LGS reg64, mem16:32</i>	0F B5/r	Load GS:reg32 with a specified far pointer from memory.

Related Instructions

LDS, LES, LFS, LSS.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and the segment is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)			X	The SS register is being loaded and the segment is marked not present.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LODS Load String

LODSB

LODSW

LODSD

LODSQ

These instructions load AL, AX, EAX, or RAX with a memory operand pointed to by DS:SI, DS:ESI, or DS:RSI. If DF = 0, rSI is incremented to point to the next location. If DF = 1, rSI is decremented to point to the previous location. rSI is incremented/decremented by 1, 2, 4, or 8 depending on the number of bytes being loaded.

The location is always specified by the DS:rSI registers, which must be loaded correctly before the load string instruction is executed.

The LODSB instruction loads AX with a byte from the memory location pointed to by DS:ESI. If DF = 0, SI will be incremented to point to the next location. If DF = 1, SI will be decremented to point to the next location. SI is incremented/decremented by 1.

The LODSW instruction loads AX with a word from the memory location pointed to by DS:ESI. If DF = 0, SI is incremented to point to the next location. If DF = 1, SI will be decremented to point to the next location. SI is incremented/decremented by 2.

The LODSD instruction loads EAX with a doubleword from the memory location pointed to by DS:ESI. If DF = 0, SI will be incremented by 4 to point to the next location. If DF = 1, SI will be decremented by 4 to point to the next location.

The LODSQ instruction loads RAX with a quadword from the memory location pointed to by DS:RSI. If DF = 0, RSI will be incremented by 8 to point to the next location. If DF = 1, RSI will be decremented by 8 to point to the next location. This mnemonic is meaningful only in 64-bit mode.

Mnemonic	Opcode	Description
LODS <i>mem8</i>	AC	Load byte at DS:SI into AL
LODS <i>mem16</i>	AD	Load word at DS:SI into AX.
LODS <i>mem32</i>	AD	Load doubleword at DS:ESI into EAX
LODS <i>mem64</i>	AD	Load quadword at DS:RSI into RAX
LODSB	AC	Load byte at DS:SI into AL
LODSW	AD	Load the word at DS:SI into AX

LODSD	AD	Load doubleword at DS:ESI into EAX
LODSQ	AD	Load quadword at DS:RSI into RAX

The forms of the LODSx instruction that have an explicit operand are equivalent to the forms with implicit operand (in fact, they have the same opcodes). The explicit forms, however, only allow specification of the operand size and an override of the default DS segment used to access the [rSI] operand. The address of the explicit operand is otherwise ignored, and instead, the implicit operand—seg:[rSI]—is always used.

The REP prefix can be used before the LODSx instructions, but more often the LODSx instruction is put inside a loop controlled by a LOOPcc instruction, as a more efficient replacement for instructions like:

```
mov eax, dword ptr ds:[esi]
add esi, 4
```

The REP prefix can be used with the LODS instruction. For details about the REP prefix, see “Repeat Prefixes” on page 10.

Related Instructions

LODSB, LODSD, LODSQ, LODSW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LOOP Loop

LOOPE

LOOPNE

LOOPNZ

LOOPZ

Decrements rCX by 1, then, if rCX not zero, jumps to the offset indicated by the operand. Otherwise continues with the next instruction below the LOOPcc instruction. This instruction is equivalent to

```
DEC    CX
JNZ    target
```

The ZF flag is unaffected by this instruction. Usually a compare instruction is used within the loop so that the ZF flag is set, and the loop execution terminates.

The target instruction is specified with a relative offset (a signed offset relative to the current value of the instruction pointer in the EIP register). This offset is generally specified as a label in the assembly code, but at the machine code level, it is encoded as a signed, 8-bit immediate value, which is added to the instruction pointer. Offset ranges of -128 to +127 are allowed.

In 64-bit mode, the operand size defaults to 64 bits without the need for a REX prefix, and the target RIP = RIP + 8-bit sign-extended displacement.

The LOOPE/LOOPZ instructions decrement rCX by 1 and then jump to location indicated by the operand if rCX and ZF are not zero. Otherwise they continue with the next instruction below the LOOPcc. In other words, the instruction exits the loop if rCX becomes 0 or ZF = 0.

The LOOPNE/LOOPNZ instructions decrement rCX by 1 and then jump to location indicated by the operand if rCX and ZF are not zero. Otherwise they continue with the next instruction below the LOOPcc. In other words, the instruction exits the loop if CX becomes 0 or ZF = 1.

Mnemonic	Opcode	Description
LOOP <i>rel8off</i>	E2 <i>cb</i>	Jump short if the CX is not zero after decrementing count.
LOOPE <i>rel8off</i>	E1 <i>cb</i>	Jump short if the CX is not zero after decrementing count and ZF is 1.
LOOPNE <i>rel8off</i>	E0 <i>cb</i>	Jump short if the CX is not zero after decrementing count and ZF is 0.

LOOPNZ <i>rel8off</i>	E0 <i>cb</i>	Jump short if the CX is not zero after decrementing count and ZF is 0.
LOOPZ <i>rel8off</i>	E1 <i>cb</i>	Jump short if the CX is not zero after decrementing count and ZF is 1.

Related Instructions

LOOPE, LOOPZ, LOOPNE, LOOPNZ.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	The offset jumped to is beyond the limits of the code segment.

LSL Load Segment Limit

The LSL instruction loads a general-purpose register with the segment limit from the segment descriptor specified by the segment selector in a 16-bit or 32-bit memory or register operand.

Mnemonic	Opcode	Description
<i>LSL reg16, reg/mem16</i>	0F 03 /r	Loads a 16-bit general-purpose register with segment limit for a selector specified in a 16-bit memory or register operand.
<i>LSL reg32, reg/mem32</i>	0F 03 /r	Loads a 32-bit general-purpose register with segment limit for a selector specified in a 32-bit memory or register operand.
<i>LSL reg364, reg/mem32</i>	0F 03 /r	Loads a 64-bit general-purpose register with segment limit for a selector specified in a 32-bit memory or register operand.

The segment limit is a 20-bit value contained in bytes 0 and 1 and in the first 4 bits of byte 6 of the segment descriptor. If the descriptor has a byte granular segment limit (the granularity flag is set to 0), the destination operand is loaded with a byte granular value (byte limit). If the descriptor has a page granular segment limit (the granularity flag is set to 1), the LSL instruction will translate the page granular limit (page limit) into a byte limit before loading it into the destination operand. The translation is performed by shifting the 20-bit “raw” limit left 12 bits and filling the low-order 12 bits with 1s.

In 64-bit mode, for both 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits.

The LSL instruction performs the following:

- Checks to make sure that the segment selector exists.
- Check whether the specified segment selector points to a descriptor that is within the limits of the global descriptor table or local descriptor table being accessed.
- Checks whether the specified segment descriptor is visible at the current privilege level (CPL)
- Checks that the descriptor type is valid for the LAR instruction. Valid descriptor types are the following:

Type	Name
1	Available 16-bit TSS
2	LDT
3	Busy 16-bit TSS
4	16-bit call gate
5	16-bit or 32-bit task gate
9	Available 32-bit TSS
B	Busy 32-bit TSS
C	32-bit call gate

If the segment selector passes these checks and the segment limit is loaded into the destination general-purpose register, the zero flag of the rFLAGS register is set to 1. If it does not, then the zero flag is cleared and the segment limit is not loaded.

The segment limit is a 20-bit value. When loading the target register, this value is converted into a 32-bit value by shifting the 20-bit value to the left 12 bits and filling the low order 12 bits with 1s.

If the granularity bit is 1, a byte limit is loaded; if the granularity is zero, a page limit is loaded.

When the operand size is 16 bits, the upper 16 bits of the 32-bit adjusted segment limit are truncated and the lower 16-bits are loaded into the target register.

ZF (bit 6 of rFLAGS) is set if the segment limit is loaded successfully, otherwise it is cleared.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
					0	0							1/0			
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

LSS**Load SS Segment Register**

Loads the contents of two memory locations indicated by the source into a destination register, and loads SS with the contents of the two succeeding memory locations. Useful for accessing an extra segment and its offset. This instruction is similar to LDS except that the SS and its offset are being loaded.

In 64-bit mode, for both 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits.

If this instruction is executed in protected mode, additional information from the segment descriptor pointed to by the segment selector in the source operand is loaded in the hidden part of the selected segment register. Any subsequent reference to a segment whose corresponding segment register is loaded with a null selector, causes a general-protection exception (#GP) and no memory reference to the segment occurs.

Mnemonic	Opcode	Description
LSS <i>reg16, mem16:16</i>	0F B2 /r	Load SS: <i>reg16</i> with a specified far pointer from memory.
LSS <i>reg32, mem16:32</i>	0F B2 /r	Load SS: <i>reg32</i> with a specified far pointer from memory.
LSS <i>reg64, mem16:32</i>	0F B2 /r	Load SS: <i>reg64</i> with a specified far pointer from memory.

Related Instructions

LSL

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The source operand is not a memory location.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and the segment is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS (selector)			X	The SS register is being loaded and the segment is marked not present.
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the segment is a nonwritable data segment, or DPL is not equal to CPL.
			X	The DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL and CPL are greater than DPL.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MFENCE**Memory Fence**

The MFENCE instruction acts as a barrier to force strong memory ordering (serialization) between load and store instructions preceding the MFENCE, and load and store instructions that follow the MFENCE. In a weakly-ordered memory system, hardware is allowed to reorder reads and writes between the processor and memory. The MFENCE instruction guarantees that all prior memory accesses complete before subsequent accesses are executed.

MFENCE is weakly-ordered with respect to data and instruction prefetches. Speculative loads initiated by the processor, or those specified explicitly using cache-prefetch instructions, can be reordered around an MFENCE.

In addition to load and store instructions, MFENCE is strongly ordered with respect to other MFENCE instructions, LFENCE instructions, SFENCE instructions, serializing instructions, and CLFLUSH instructions.

Support for the MFENCE instruction is indicated when the SSE2 bit (bit 26) is set to 1 in EDX after executing CPUID with extended function 8000_0001h.

Mnemonic	Opcode	Description
MFENCE <i>mmx1, mmx2/mem64</i>	OF AE F0	Force strong ordering of (serialize) load and store operations.

Related Instructions

LFENCE, SFENCE

rFLAGS Affected

None.

Exceptions

None.

MOV**Move**

Copies a byte, word, doubleword, or quadword from a general-purpose register, segment register, memory location, or an immediate number to a register or memory location. The source and destination must be of the same size and cannot both be memory locations.

In opcodes A0 through A3, memory offsets are address-sized. In 64-bit mode, memory offsets (called *moffsets*) default to 64 bits. A REX prefix is not needed to specify the 64-bit memory offset. The B8 through BF opcodes, in 64-bit mode, are the only cases in which a 64-bit immediate value can be used (in all other cases, immediate values are a maximum of 32 bits in 64-bit mode).

For segment-register moves in legacy mode with a 32-bit operand size, 16-bit register results are zero-extended to 32 bits. For segment-register moves in 64-bit mode with 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits. If the destination operand is a segment register (DS, ES, FS, GS, or SS), the source operand must be a valid segment selector.

Loading the SS register with a MOV instruction inhibits all interrupts until after the execution of the next instruction. This operation allows a stack pointer to be loaded into the ESP register with the next instruction (MOV ESP, stack-pointer value) before an interrupt occurs ¹. The LSS instruction offers a more efficient method of loading the SS and ESP registers.

The MOV instruction cannot be used to load the CS register. Attempting to do so results in an invalid opcode exception (#UD). To load the CS register, use the far JMP, CALL, or RET instruction.

To initialize a register to 0, rather than using a MOV instruction it may be more efficient to use the XOR instruction with identical destination and source operands.

Mnemonic	Opcode	Description
MOV <i>reg/mem8, reg8</i>	88/ <i>r</i>	Move the contents of an 8-bit register to an 8-bit destination register or memory operand.
MOV <i>reg/mem16, reg16</i>	89/ <i>r</i>	Move the contents of a 16-bit register to a 16-bit destination register or memory operand.
MOV <i>reg/mem32, reg32</i>	89/ <i>r</i>	Move the contents of a 32-bit register to a 32-bit destination register or memory operand.
MOV <i>reg/mem64, reg64</i>	89/ <i>r</i>	Move the contents of a 64-bit register to a 64-bit destination register or memory operand.

Mnemonic	Opcode	Description
MOV <i>reg8, reg/mem8</i>	8A/ <i>r</i>	Move the contents of an 8-bit register or memory operand to an 8-bit destination register.
MOV <i>reg16, reg/mem16</i>	8B/ <i>r</i>	Move the contents of a 16-bit register or memory operand to a 16-bit destination register.
MOV <i>reg32, reg/mem32</i>	8B/ <i>r</i>	Move the contents of a 32-bit register or memory operand to a 32-bit destination register.
MOV <i>reg64, reg/mem64</i>	8B/ <i>r</i>	Move the contents of a 64-bit register or memory operand to a 64-bit destination register.
MOV <i>reg16/32/64/mem16, segReg</i>	8C/ <i>r</i>	Move the contents of a segment register to a 16-bit, 32-bit, or 64-bit destination register or to a 16-bit memory operand.
MOV <i>segReg, reg16/32/64/mem16</i>	8E/ <i>r</i>	Move the low 16-bits of a 16-bit, 32-bit, or 64-bit register or of a 16-bit memory operand to a segment register.
MOV AL, <i>moffset8</i>	A0	Move 8-bit data at a specified memory offset to the AL register.
MOV AX, <i>moffset16</i>	A1	Move 16-bit data at a specified memory offset to the AX register.
MOV EAX, <i>moffset32</i>	A1	Move 32-bit data at a specified memory offset to the EAX register.
MOV RAX, <i>moffset64</i>	A1	Move 64-bit data at a specified memory offset to the RAX register.
MOV <i>moffset8</i> , AL	A2	Move the contents of the AL register to an 8-bit memory offset.
MOV <i>moffset16</i> , AX	A3	Move the contents of the AX register to a 16-bit memory offset.
MOV <i>moffset32</i> , EAX	A3	Move the contents of the EAX register to a 32-bit memory offset.
MOV <i>moffset64</i> , RAX	A3	Move the contents of the RAX register to a 64-bit memory offset.
MOV <i>reg8, imm8</i>	B + <i>rb</i>	Move an 8-bit immediate value into an 8-bit register.
MOV <i>reg16, imm16</i>	B8 + <i>rw</i>	Move a 16-bit immediate value into a 16-bit register.
MOV <i>reg32, imm32</i>	B8 + <i>rd</i>	Move an 32-bit immediate value into a 32-bit register.
MOV <i>reg64, imm64</i>	B8 + <i>rq</i>	Move an 64-bit immediate value into a 64-bit register.
MOV <i>reg/mem8, imm8</i>	C6 /0	Move an 8-bit immediate value to an 8-bit register or memory operand.
MOV <i>reg/mem16, imm16</i>	C7 /0	Move a 16-bit immediate value to a 16-bit register or memory operand.

Mnemonic	Opcode	Description
<i>MOV reg/mem32, imm32</i>	<i>C7 /0</i>	Move a 32-bit immediate value to a 32-bit register or memory operand.
<i>MOV reg/mem64, imm32</i>	<i>C7 /0</i>	Move a 32-bit immediate value to a 64-bit register or memory operand.

Related Instructions

MOV(2), MOV(3), MOVD, MOVS, MOVSB, MOVSD, MOVSW, MOVSX, MOVZX

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	An attempt was made to load the CS register.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)			X	The SS register is being loaded and the segment pointed to is marked not present.
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The null selector is loaded into the SS register.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.

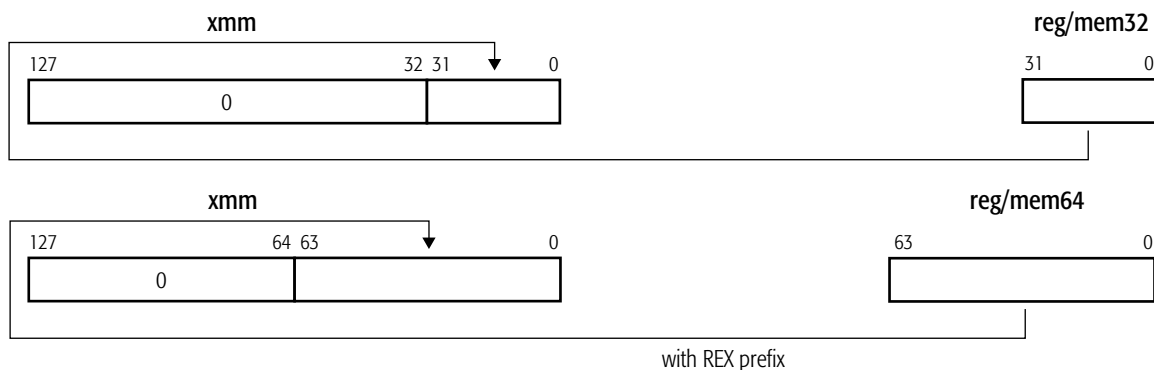
Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP (selector)			X	The segment selector index is outside the descriptor table limits.
			X	The SS register is being loaded and the segment selector RPL and the segment descriptor DPL are not equal to the CPL.
			X	The SS register is being loaded and the segment pointed to is a non-writable data segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MOVD**Move Doubleword or Quadword**

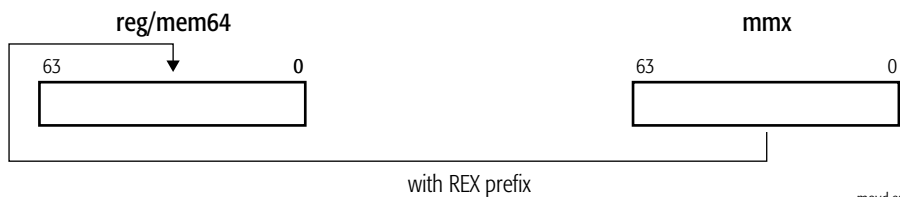
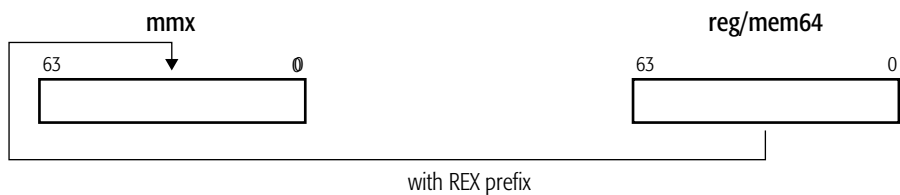
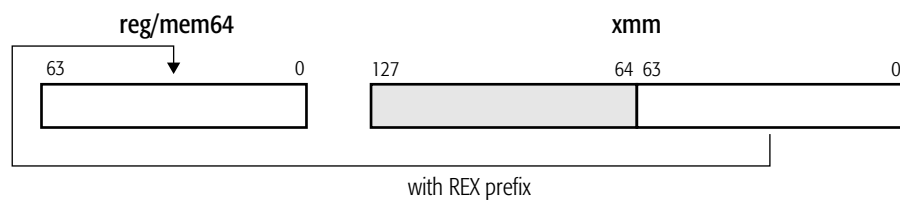
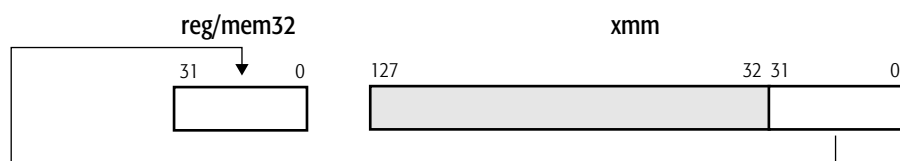
The MOVD instruction moves a 32-bit or 64-bit value:

- from a 32-bit or 64-bit general-purpose register or memory location to the low-order 32 or 64 bits of an XMM register, with zero-extension to 128 bits, or
- from the low-order 32 or 64 bits of an XMM to a 32-bit or 64-bit general-purpose register or memory location, or
- from a 32-bit or 64-bit general-purpose register or memory location to the low-order 32 bits (with zero-extension to 64 bits) or the full 64 bits of an MMX register, or
- from the low-order 32 or the full 64 bits of an MMX register to a 32-bit or 64-bit general-purpose register or memory location.

Mnemonic	Opcode	Description
MOVD <i>xmm, reg/mem32</i>	66 0F 6E/ <i>r</i>	Moves 32-bit value from a general-purpose register or 32-bit memory location to an XMM register.
MOVD <i>xmm, reg/mem64</i>	66 0F 6E/ <i>r</i>	Moves 64-bit value from a general-purpose register or 64-bit memory location to an XMM register.
MOVD <i>reg/mem32, xmm</i>	66 0F 7E/ <i>r</i>	Moves 32-bit value from an XMM register to a general-purpose register or 32-bit memory location.
MOVD <i>reg/mem64, xmm</i>	66 0F 7E/ <i>r</i>	Moves 64-bit value from an XMM register to a general-purpose register or 64-bit memory location.
MOVD <i>mmx, reg/mem32</i>	0F 6E/ <i>r</i>	Moves 32-bit value from a general-purpose register or 32-bit memory location to an MMX register.
MOVD <i>mmx, reg/mem64</i>	0F 6E/ <i>r</i>	Moves 64-bit value from a general-purpose register or 64-bit memory location to an MMX register.
MOVD <i>reg/mem32, mmx</i>	0F 7E/ <i>r</i>	Moves 32-bit value from an MMX register to a general-purpose register or 32-bit memory location.
MOVD <i>reg/mem64, mmx</i>	0F 7E/ <i>r</i>	Moves 64-bit value from an MMX register to a general-purpose register or 64-bit memory location.



All operations are "copy"



movd.eps

Related Instructions

MOVDQA, MOVDQU, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

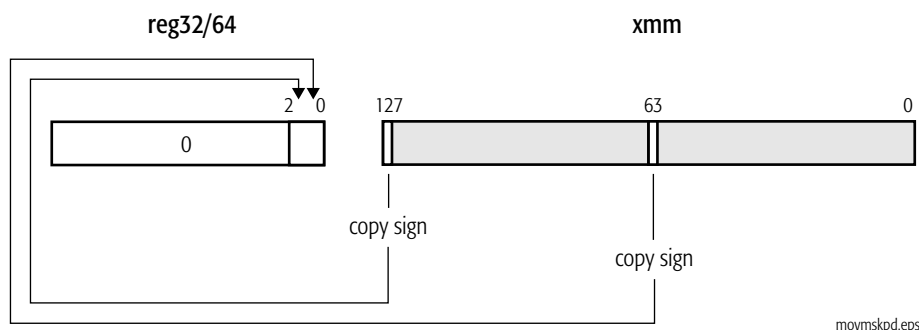
Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the EFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVMSKPD**Extract Packed Double-Precision Floating-Point Sign Mask**

The MOVMSKPD instruction moves the sign bits of two packed double-precision floating-point values in an XMM register to the two low-order bits of a general-purpose register, with zero-extension.

Mnemonic	Opcode	Description
MOVMSKPD <i>reg32, xmm</i>	66 0F 50 /r	Move sign bits in an XMM register to a 32-bit general-purpose register.
MOVMSKPD <i>reg64, xmm</i>	66 0F 50 /r	Move sign bits in an XMM register to a 64-bit general-purpose register.

**Related Instructions**

MOVMSKPS, PMOVMSKB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

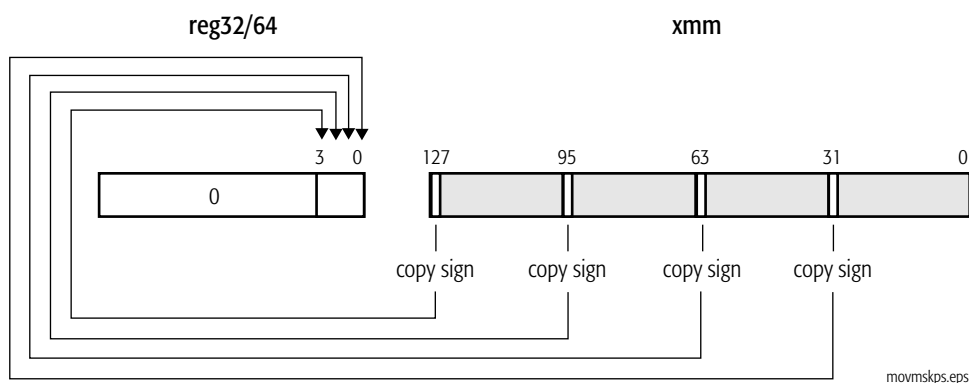
Exceptions

Exception (vector)	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

MOVMSKPS**Extract Packed Single-Precision Floating-Point Sign Mask**

The MOVMSKPD instruction moves the sign bits of four packed single-precision floating-point values in an XMM register to the four low-order bits of a general-purpose register, with zero-extension.

Mnemonic	Opcode	Description
MOVMSKPS <i>reg32, xmm</i>	0F 50 /r	Move sign bits in an XMM register to a 32-bit general-purpose register.
MOVMSKPS <i>reg64, xmm</i>	0F 50 /r	Move sign bits in an XMM register to a 64-bit general-purpose register.

**Related Instructions**

MOVMSKPD, PMOVMSKB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

MOVNTI Move Non-Temporal Doubleword or Quadword

The MOVNTI instruction stores a 32-bit or 64-bit general-purpose register value into a memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see Section 3.9, “Memory Optimization”, on page 115 of Volume 1.

MOVNTI is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTI with respect to other stores.

Support for the MOVNTI instruction is indicated when the SSE2 bit (bit 26) is set to 1 in EDX after executing CPUID with extended function 8000_0001h.

Mnemonic	Opcode	Description
MOVNTI <i>mem32, reg32</i>	0F C3 /r	Stores a 32-bit general-purpose register value into a 32-bit memory location, minimizing cache pollution.
MOVNTI <i>mem64, reg64</i>	0F C3 /r	Stores a 64-bit general-purpose register value into a 64-bit memory location, minimizing cache pollution.

Related Instructions

MOVNTDQ, MOVNTPD, MOVNTPS, MOVNTQ

Exceptions

Exception (vector)	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVS **Move String**

MOVSB

MOVSW

MOVSD

MOVSQ

Moves a byte, word, doubleword, or quadword from memory location pointed to by DS:SI (byte or word), DS:ESI (word), or DS:RSI (quadword) to the memory location pointed to by ES:DI (byte or word), ES:EDI (word), or ES:RDI (quadword).

If DF = 0, both pointers are incremented; otherwise, they are decremented. SI and DI are incremented/decremented by 1, 2, 4, or 8, depending on whether it is a byte, word, doubleword, or quadword string.

The DS segment may be overridden with a segment override prefix, but the ES segment cannot be overridden.

The MOVSB instruction moves a byte from the memory location pointed to by DS:SI to the memory location pointed to by ES:DI. If DF = 0, both pointers are incremented; otherwise, they are decremented. SI and DI are incremented or decremented by one.

The MOVSW instruction moves a word from the memory location pointed to by DS:SI to memory location pointed to by ES:DI. If DF = 0, both pointers are incremented; otherwise, they are decremented. SI and DI are incremented/decremented by 2.

The MOVSD instruction moves the doubleword from the memory location pointed to by DS:ESI to the memory location pointed to by ES:EDI. If DF = 0, both pointers are incremented; otherwise, they are decremented. ESI and EDI are incremented or decremented by 4. This MOVSD instruction should not be confused with the same-mnemonic MOVSD (move scalar double-precision floating-point) instruction in the 128-bit media instruction set. Assemblers can distinguish the instructions by the number and type of operands.

The MOVSQ instruction moves the quadword from the memory location pointed to by DS:RSI to the memory location pointed to by ES:RDI. If DF = 0, both pointers are incremented; otherwise, they are decremented. RSI and RDI are incremented or decremented by 8. This mnemonic is meaningful only in 64-bit mode.

Mnemonic	Opcode	Description
MOVS <i>mem8, mem8</i>	A4	Move byte at DS:SI to ES:DI.
MOVS <i>mem16, mem16</i>	A5	Move word at DS:SI to ES:DI.
MOVS <i>mem32, mem32</i>	A5	Move doubleword at DS:ESI to ES:EDI.
MOVS <i>mem64, mem64</i>	A5	Move quadword at DS:RSI to ES:RDI.
MOVSB	A4	Move byte at DS:SI to ES:DI.
MOVSW	A5	Move word at DS:SI to ES:DI.
MOVSD	A5	Move doubleword at DS:ESI to ES:EDI.
MOVSQ	A5	Move quadword at DS:RSI to ES:RDI.

The forms of the MOV_Sx instruction that have explicit operands are equivalent to the forms with implicit operand (in fact, they have the same opcodes). The explicit forms, however, only allow specification of the operand size and an override of the default DS segment used to access the [rSI] operand. The addresses of these explicit operands are otherwise ignored, and instead, the implicit operands—seg:[rSI] and ES:[rDI]—are always used.

The REP prefix can be used with the MOV_Sx instructions. For details about the REP prefix, see “Repeat Prefixes” on page 10. When used with the REP prefix, CX is decremented each iteration until it reaches zero. The MOV_Sx instructions can also be put inside a LOOPcc instruction.

Related Instructions

MOVSB, MOVSD, MOVSQ, MOVSW.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MOVSX Move with Sign-Extension

Extends the MSB of an 8-bit or 16-bit operand into all higher bits in a 16-bit, 32-bit, or 64-bit register.

Mnemonic	Opcode	Description
<i>MOVSX reg16, reg/mem8</i>	0F BE /r	Move the contents of an 8-bit register or memory operand to a 16-bit register with sign extension.
<i>MOVSX reg32, reg/mem8</i>	0F BE /r	Move the contents of an 8-bit register or memory operand to a 32-bit register with sign extension.
<i>MOVSX reg64, reg/mem8</i>	0F BE /r	Move the contents of an 8-bit register or memory operand to a 64-bit register with sign extension.
<i>MOVSX reg32, reg/mem16</i>	0F BF /r	Move the contents of an 16-bit register or memory operand to a 32-bit register with sign extension.
<i>MOVSX reg64, reg/mem16</i>	0F BF /r	Move the contents of an 16-bit register or memory operand to a 64-bit register with sign extension.

Related Instructions

MOVSXD, MOV, MOVS, MOVSB, MOVSD, MOVSQ, MOVSW, MOVZX

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MOVSXD Move with Sign-Extend Doubleword

Extends a signed 32-bit operand into 64-bits. The source operand is sign extended to 64 bits, and is stored in the 64-bit general purpose register.

A REX prefix is needed to do useful work with this instruction—the work of sign-extending a 32-bit source operand to a 64-bit result. Without a REX prefix, the operand size is 32 bits and the source is zero-extended into a 64-bit register. With a 16-bit operand size, the instruction is a NOP.

This instruction is available only in 64-bit mode. An invalid-opcode exception occurs if this instruction is used in legacy or compatibility mode.

Mnemonic	Opcode	Description
MOVSXD <i>reg64, reg/mem32</i>	63 /r	Move the contents of a 32-bit register or memory operand to a 64-bit register with sign extension.

Related Instructions

MOVSX, MOV, MOVS, MOVSB, MOVSD, MOVSQ, MOVSW, MOVZX

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in legacy or compatibility mode.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MOVZX Move with Zero-Extension

Copies the contents of the source operand (register or memory location) to the destination operand (register) and zero extends (writes a zero to each bit) the value to 16, 32, or 64 bit register. The size of the converted value depends on the operand-size attribute.

This instruction is used in unsigned arithmetic operations.

Mnemonic	Opcode	Description
MOVZX <i>reg16, reg/mem8</i>	0F B6 /r	Move the contents of an 8-bit register or memory operand to a 16-bit register with zero-extension.
MOVZX <i>reg32, reg/mem8</i>	0F B6 /r	Move the contents of an 8-bit register or memory operand to a 32-bit register with zero-extension.
MOVZX <i>reg64, reg/mem8</i>	0F B6 /r	Move the contents of an 8-bit register or memory operand to a 64-bit register with zero-extension.
MOVZX <i>reg32, reg/mem16</i>	0F B7 /r	Move the contents of an 16-bit register or memory operand to a 32-bit register with zero-extension.
MOVZX <i>reg64, reg/mem16</i>	0F B7 /r	Move the contents of an 16-bit register or memory operand to a 64-bit register with zero-extension.

Related Instructions

MOV, MOVS, MOVSB, MOVSD, MOVSXD, MOVSW, MOVZX

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

MUL Multiply Unsigned

Multiplies an unsigned byte, word, doubleword, or quadword indicated by the operand by a value in AL, AX, EAX, or RAX.

The result is stored in register AX, register pair DX:AX, or register pair EDX:EAX or RDX:RAX (depending on the operand size), with the high-order bits of the product contained in register AH, DX, EDX, or RDX, respectively. The carry flag (CF) and overflow flag (OF) are set to 1 if the upper half of the product is non-zero. Otherwise, CF and OF are cleared to 0. The other arithmetic flags (SF, ZF, AF, PF) are undefined.

Mnemonic	Opcode	Description
MUL <i>reg/mem8</i>	F6 /4	Multiplies a 8-bit register or memory operand by the contents of the AL register and stores the result in the AX register.
MUL <i>reg/mem16</i>	F7 /4	Multiplies a 16-bit register or memory operand by the contents of the AX register and stores the result in the DX:AX register.
MUL <i>reg/mem32</i>	F7 /4	Multiplies a 32-bit register or memory operand by the contents of the EAX register and stores the result in the EDX:EAX register.
MUL <i>reg/mem64</i>	F7 /4	Multiplies a 64-bit register or memory operand by the contents of the RAX register and stores the result in the RDX:RAX register.

Related Instructions

DIV, FIDIV, FIMUL.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				U	U	U	U	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

NEG

Two's Complement Negation

Performs two's complement of operand. (This operation is equivalent to subtracting the operand from 0.) The destination operand is located in a general-purpose register or a memory location. This instruction should only be used on signed integer numbers.

The CF flag cleared to 0 if the source operand is 0; otherwise it is set to 1. The OF, SF, ZF, AF, and PF flags are set according to the result.

The LOCK prefix can be used with forms of the NEG instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
NEG <i>reg/mem8</i>	F6 /3	Performs a two's complement negation on an 8-bit register or memory operand.
NEG <i>reg/mem16</i>	F7 /3	Performs a two's complement negation on a 16-bit register or memory operand.
NEG <i>reg/mem32</i>	F7 /3	Performs a two's complement negation on a 32-bit register or memory operand.
NEG <i>reg/mem64</i>	F7 /3	Performs a two's complement negation on a 64-bit register or memory operand.

Related Instructions

AND, NOT, OR, XOR.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

NOP **No Operation**

Performs no operation. Sometimes used to account for timing delays.

This instruction is a one-byte instruction that takes up space in the instruction stream but does not affect the machine context, except the EIP register. Updates EIP to point to next instruction following NOP.

The NOP instruction is an alias mnemonic for the XCHG eAX, eAX instruction.

Mnemonic	Opcode	Description
NOP	90	Perform no operation.

Related Instructions

FNOP.

rFLAGS Affected

None.

Exceptions

None.

NOT One's Complement Negation

Replaces the operand with its negation (the ones complement). Each bit is inverted. The destination operand can be a register or a memory location.

The LOCK prefix can be used with forms of the NOT instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
NOT <i>reg/mem8</i>	F6 /2	Reverses the bits in an 8-bit register or memory operand.
NOT <i>reg/mem16</i>	F7 /2	Reverses the bits in a 16-bit register or memory operand.
NOT <i>reg/mem32</i>	F7 /2	Reverses the bits in a 32-bit register or memory operand.
NOT <i>reg/mem64</i>	F7 /2	Reverses the bits in a 64-bit register or memory operand.

Related Instructions

AND, NEG, OR, XOR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

OR Logical OR

Performs logical OR on the bits of two operands, replacing the destination operand with the result. This instruction is useful for turning a bit on or testing for a zero value in a register.

The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result of the OR instruction is 0 if both corresponding bits of the operands are 0; otherwise, each bit is 1.

The LOCK prefix can be used with forms of the OR instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
OR AL, <i>imm8</i>	0C <i>ib</i>	Performs a logical OR of AL and an immediate 8-bit value.
OR AX, <i>imm16</i>	0D <i>iw</i>	Performs a logical OR of AX and an immediate 16-bit value.
OR EAX, <i>imm32</i>	0D <i>id</i>	Performs a logical OR of EAX and an immediate 32-bit value.
OR RAX, <i>imm32</i>	0D <i>id</i>	Performs a logical OR of RAX and an immediate 32-bit value.
OR <i>reg/mem8</i> , <i>imm8</i>	80 /1 <i>ib</i>	Performs a logical OR of the contents of an 8-bit register or memory operand and an immediate 8-bit value.
OR <i>reg/mem16</i> , <i>imm16</i>	81 /1 <i>iw</i>	Performs a logical OR of the contents of an 16-bit register or memory operand and an immediate 16-bit value.
OR <i>reg/mem32</i> , <i>imm32</i>	81 /1 <i>id</i>	Performs a logical OR of the contents of an 32-bit register or memory operand and an immediate 32-bit value.
OR <i>reg/mem64</i> , <i>imm32</i>	81 /1 <i>id</i>	Performs a logical OR of the contents of an 64-bit register or memory operand and an immediate 32-bit value.
OR <i>reg/mem16</i> , <i>imm8</i>	83 /1 <i>ib</i>	Performs a logical OR of the contents of an 16-bit register or memory operand and a sign-extended immediate 8-bit value.

Mnemonic	Opcode	Description
OR <i>reg/mem32, imm8</i>	83 /1 <i>ib</i>	Performs a logical OR of the contents of an 32-bit register or memory operand and a sign-extended immediate 8-bit value.
OR <i>reg/mem64, imm8</i>	83 /1 <i>ib</i>	Performs a logical OR of the contents of an 64-bit register or memory operand and a sign-extended immediate 8-bit value.
OR <i>reg/mem8, reg8</i>	08 / <i>r</i>	Perform a logical OR of the contents of an 8-bit register or memory operand with the contents of an 8-bit register.
OR <i>reg/mem16, reg16</i>	09 / <i>r</i>	Perform a logical OR of the contents of a 16-bit register or memory operand with the contents of a 16-bit register.
OR <i>reg/mem32, reg32</i>	09 / <i>r</i>	Perform a logical OR of the contents of a 32-bit register or memory operand with the contents of a 32-bit register.
OR <i>reg/mem64, reg64</i>	09 / <i>r</i>	Perform a logical OR of the contents of a 64-bit register or memory operand with the contents of a 64-bit register.
OR <i>reg8, reg/mem8</i>	0A / <i>r</i>	Performs a logical OR of the contents of an 8-bit register with the contents of an 8-bit register or memory operand.
OR <i>reg16, reg/mem16</i>	0B / <i>r</i>	Performs a logical OR of the contents of a 16-bit register with the contents of a 16-bit register or memory operand.
OR <i>reg32, reg/mem32</i>	0B / <i>r</i>	Performs a logical OR of the contents of a 32-bit register with the contents of a 32-bit register or memory operand.
OR <i>reg64, reg/mem64</i>	0B / <i>r</i>	Performs a logical OR of the contents of a 64-bit register with the contents of a 64-bit register or memory operand.

The following chart summarizes the affect of this instruction:

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Related Instructions

AND, NEG, NOT, XOR

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				1/0	1/0	U	1/0	0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X X X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). The destination operand is in a nonwritable segment. There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

OUT Output to Port

Copies the value from source operand to the I/O port specified with the destination operand. The source operand can be register AL, AX, or EAX, depending on the size of the port being accessed (8, 16, or 32 bits, respectively); the destination operand can be a byte-immediate or the DX register.

Using a byte immediate allows I/O port addresses 0 to 255 to be accessed; using the DX register as a source operand allows I/O ports from 0 to 65,535 to be accessed.

Mnemonic	Opcode	Description
OUT <i>imm8</i> , AL	E6 <i>ib</i>	Output the byte in the AL register to the port specified by an 8-bit immediate value.
OUT <i>imm8</i> , AX	E7 <i>ib</i>	Output the word in the AX register to the port specified by an 8-bit immediate value.
OUT <i>imm8</i> , EAX	E7 <i>ib</i>	Output the doubleword in the EAX register to the port specified by an 8-bit immediate value.
OUT DX, AL	EE	Output byte in AL to the output port specified in DX.
OUT DX, AX	EF	Output byte in AX to the output port specified in DX.
OUT DX, EAX	EF	Output byte in EAX to the output port specified in DX.

Related Instructions

IN.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	At least one of the corresponding I/O permission bits is 1 in the TSS for the accessed I/O port. The current privilege level is greater than the I/O privilege level of the current task and at least one of the corresponding I/O permission bits is 1 in the TSS for the accessed I/O port.

The forms of the OUTSx instructions that have explicit operands are only used by the assembler to choose a segment register and address size attribute. The explicit memory operand (the second operand) allows specification of the operand size and, specification of an override to the default DS prefix used to access the [eSI] operand. The memory operand is otherwise ignored, and instead, the implicit operand, *seg:[eSI]*, is always used. The explicit register operand (the first operand) specifies the I/O port address and must always be DX.

Related Instructions

INS, OUTSB, OUTSD, OUTSW.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X			During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	At least one of the corresponding I/O permission bits is 1 in the TSS for the accessed I/O port.
			X	The segment register contains a null segment selector.
			X	The current privilege level is greater than the I/O privilege level of the current task and at least one of the corresponding I/O permission bits is 1 in the TSS for the accessed I/O port.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

POP**Pop Stack**

Copies the word pointed to by the stack pointer to the register or memory location indicated by the operand and increments the SP by 2 for the 16-bit stack only.

The address-size attribute of the stack segment determines the stack pointer size (16 bits or 32 bits—the source address size), and the operand-size attribute of the current code segment determines the amount the stack pointer is incremented (2, 4 or 8 bytes). For example, if these address- and operand-size attributes are 32, the 32-bit ESP register (stack pointer) is incremented by 4 and, if they are 16, the 16-bit SP register is incremented by 2.

In 64-bit mode, the operand size defaults to 64 bits, and there is no prefix available for encoding a 32-bit operand size. An invalid-opcode exception occurs if the 1F, 07, or 17 opcode of this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
POP <i>reg/mem16</i>	8F /0	Pop the top of the stack into a register or 16-bit memory location and increment the stack pointer.
POP <i>reg/mem32</i>	8F /0	Pop the top of the stack into a register or 32-bit memory location and increment the stack pointer. (No prefix for encoding this in 64-bit mode.)
POP <i>reg/mem64</i>	8F /0	Pop the top of the stack into a register or 64-bit memory location and increment the stack pointer.
POP <i>reg16</i>	58 + <i>rw</i>	Pop the top of the stack into a 16-bit register and increment the stack pointer.
POP <i>reg32</i>	58 + <i>rd</i>	Pop the top of the stack into a 32-bit register and increment the stack pointer. (No prefix for encoding this in 64-bit mode.)
POP <i>reg64</i>	58 + <i>rq</i>	Pop the top of the stack into a 64-bit register and increment the stack pointer.
POP DS	1F	Pop the top of the stack into the DS register and increment the stack pointer. (Invalid in 64-bit mode.)
POP ES	07	Pop the top of the stack into the ES register and increment the stack pointer. (Invalid in 64-bit mode.)
POP SS	17	Pop the top of the stack into the SS register and increment the stack pointer. (Invalid in 64-bit mode.)

POP FS	0F A1	Pop the top of the stack into the FS register and increment the stack pointer.
POP GS	0F A9	Pop the top of the stack into the GS register and increment the stack pointer.

Related Instructions

FPOP, FPUSH, PUSH.

rFLAGS Affected

None.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	The 1F, 07, and 17 opcodes of this instruction are not recognized in 64-bit mode.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
Stack, #SS			X	The current top of the stack exceeded the stack segment limit.
			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)				The SS register is being loaded and the segment pointed to is marked not present.
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	An attempt is made to load SS with the null segment selector.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The segment selector index is outside the descriptor table limits.
			X	The SS register is being loaded and the segment selector RPL and the segment descriptor DPL are not equal to the CPL.
			X	The SS register is being loaded and the segment pointed to is a non-writable data segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

POPA POPAD

POP All to GPR

Pops words from the stack into the general-purpose registers.

The GP registers are loaded in the following order: DI, SI, BP, BX, DX, CX, and AX. The value on the stack for the ESP or SP register is ignored. Instead, the ESP or SP register is incremented after each register is loaded.

The POPAD instruction for use when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when POPA is used and to 32 when POPAD is used (using the operand-size override prefix [66H] if necessary). Others may use the current setting of the operand-size attribute to determine the size of values to be popped from the stack, regardless of the mnemonic used. (The D flag in the current code segment descriptor determines the operand-size attribute.)

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
POPA	61	Pops the DI, SI, BP, SP, BX, DX, CX, and AX registers. (Invalid in 64-bit mode.)
POPAD	61	Pops the EDI, ESI, EBP, ESP, EBX, EDX, ECX, and EAX registers. (Invalid in 64-bit mode.)

Related Instructions

FPOP, FPUSH, PUSH.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	The 1F, 07, and 17 opcodes of this instruction are not recognized in 64-bit mode.
Segment not present, #NP (selector)			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
Stack, #SS			X	The current top of the stack exceeded the stack segment limit.
			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)				The SS register is being loaded and the segment pointed to is marked not present.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	An attempt is made to load SS with the null segment selector.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, #GP (selector)			X	The segment selector index is outside the descriptor table limits.
			X	The SS register is being loaded and the segment selector RPL and the segment descriptor DPL are not equal to the CPL.
			X	The SS register is being loaded and the segment pointed to is a non-writable data segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
			X	The DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

POPF

POPF to rFLAGS

POPFQ

Copies bits previously pushed onto the stack with the PUSHF/PUSHFD/PUSHFQ instruction into the flags register and increments the stack pointer by 2, 4 or 8.

The effect of the POPF_x instruction on the rFLAGS register depends on the mode of operation of the processor. When the processor is operating in protected mode at privilege level 0 (or in real-address mode, which is equivalent to privilege level 0), all the non-reserved flags in the rFLAGS register except the VIP, VIF, and VM flags can be modified. The VIP, VIF, and RF flags are cleared, and the VM flag is unaffected.

When operating in protected mode, with a privilege level greater than 0, but less than or equal to IOPL, all the flags can be modified except the IOPL field and the VIP, VIF, and VM flags. Here, the IOPL flags are unaffected, the VIP and VIF flags are cleared, and the VM flag is unaffected.

When operating in virtual-8086 mode, if system software has set the IOPL field to a value less than 3, a general-protection exception occurs if application software attempts to execute a POPF, POPFD, POPFQ, PUSHF, PUSHFD, or PUSHFQ instruction.

Mnemonic	Opcode	Description
POPF	9D	Pop the stack into the FLAGS register (the lower 16-bits of the EFLAGS register).
POPFD	9D	Pops a double word from the stack top to the EFLAGS register and increments SP by 2.
POPFQ	9D	Pops a quadword from the stack top to the RFLAGS register and increments RSP by 8.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

POPF_START:

```
IF (REAL_MODE)
    POPF_REAL
ELSIF (PROTECTED_MODE)
    POPF_PROTECTED
ELSE // (VIRTUAL_MODE)
    POPF_VIRTUAL
```

POPF_REAL:

```

    POP.v temp_RFLAGS
    RFLAGS.v = temp_RFLAGS          // VIF,VIP,VM unchanged
                                     // RF cleared
    EXIT

```

POPF_PROTECTED:

```

    POP.v temp_RFLAGS
    RFLAGS.v = temp_RFLAGS          // VIF,VIP,VM unchanged
                                     // IOPL changed only if (CPL=0)
                                     // IF changed only if (CPL<=old_RFLAGS.IOPL)
                                     // RF cleared
    EXIT

```

POPF_VIRTUAL:

```

    IF (RFLAGS.IOPL=3)
    {
        POP.v temp_RFLAGS
        RFLAGS.v = temp_RFLAGS      // VIF,VIP,VM,IOPL unchanged
                                     // RF cleared
        EXIT
    }
    ELIF ((CR4.VME=1) && (OPERAND_SIZE=16))
    {
        POP.w temp_RFLAGS
        IF (((temp_RFLAGS.IF=1) && (RFLAGS.VIP=1)) || (temp_RFLAGS.TF=1))
            EXCEPTION [#GP(0)]
                                     // notify the virtual-mode-manager to deliver
                                     // the task's pending interrupts
        RFLAGS.w = temp_RFLAGS      // IF,IOPL unchanged
                                     // RFLAGS.VIF=temp_RFLAGS.IF
                                     // RF cleared
        EXIT
    }
    ELSE // (((RFLAGS.IOPL<3) && ((CR4.VME=0) || (OPERAND_SIZE!=16)))
        EXCEPTION [#GP(0)]

```


Related Instructions

POP, POPA, POPAD, PUSHF, PUSHFD, PUSHFQ

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
1/0			1/0		0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	The top of the stack is not within the stack segment.
General protection, #GP		X X		The I/O privilege level is less than 3. An attempt is made to execute this instruction with an operand-size override prefix.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

PREFETCH PREFETCHW

Prefetch L1 Data-Cache Line

PREFETCH and PREFETCHW are 3DNow!™ instructions. They load a cache line into the L1 data cache from the memory address specified by the *mem8* immediate value. The PREFETCH instruction loads a cache line even if the *mem8* address is not aligned with the start of the line. If a cache hit occurs, or if a memory fault is detected, no bus cycle is initiated, and the instruction is treated as a NOP.

The PREFETCHW instruction loads the prefetched line and sets the cache-line state to Modified, in anticipation of subsequent data writes to the line. The PREFETCH instruction, by contrast, typically (depending on hardware implementation) sets the cache-line state to Exclusive.

Mnemonic	Opcode	Description
PREFETCH <i>mem8</i>	0F 0D /0	Prefetch processor cache line into L1 data cache.
PREFETCHW <i>mem8</i>	0F 0D /1	Prefetch processor cache line into L1 data cache.

The opcodes for the instructions include the ModRM byte, and only the memory form of ModRM is valid. The register form of ModRM causes an invalid-opcode exception. Because there is no destination register, the three destination register field bits of the ModRM byte are used to define the type of prefetch to be performed. The PREFETCH and PREFETCHW instructions are defined by the bit pattern 000b and 001b, respectively. All other bit patterns are reserved for future use.

Table 2-3 summarizes the PREFETCH type options. The *reserved* PREFETCH types do not result in an invalid-opcode exception if executed. Instead, for forward compatibility with future processors that may implement additional forms of the PREFETCH instruction, all reserved PREFETCH types are implemented as synonyms of the basic PREFETCH type (the PREFETCH instruction with type 000b).

Table 2-3. Summary of PREFETCH Instruction Type Options

ModRM Byte	Result ¹
11-xxx-xxx	Invalid opcode
mm-000-xxx	PREFETCH
mm-001-xxx	PREFETCHW
mm-010-xxx	<i>reserved</i>
mm-011-xxx	<i>reserved</i>
mm-100-xxx	<i>reserved</i>
mm-101-xxx	<i>reserved</i>
mm-110-xxx	<i>reserved</i>
mm-111-xxx	<i>reserved</i>
Note: 1. The “Reserved” PREFETCH types do not result in an Invalid Opcode Exception if executed. Instead, for forward compatibility with future processors that may implement additional forms of the PREFETCH instruction, all “Reserved” PREFETCH types are implemented as synonyms for the basic PREFETCH type (the PREFETCH instruction with type 000b)	

The operation of these instructions is implementation-dependent. The instructions can be ignored or changed by a processor implementation. The size of the cache line also depends on the implementation, with a minimum size of 32 bytes. For details on the use of this instruction, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

Related Instructions

PREFETCH^{level}

rFLAGS Affected

None.

Exceptions

Exception (vector)	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.

PREFETCH_{level}**Prefetch Data to Cache Level _{level}**

The **PREFETCH_{level}** instruction loads a cache line, from the memory address specified by the *mem8* immediate value, into the data-cache level specified by the locality reference bits 5:3 of the ModRM byte. This instruction loads a cache line even if the *mem8* address is not aligned with the start of the line. If the cache line is already contained in a cache level that is lower than the specified locality reference, or if a memory fault is detected, a bus cycle is not initiated, and the instruction is treated as a NOP.

Mnemonic	Opcode	Description
PREFETCHNTA <i>mem8</i>	0F 18 /0	Move data closer to the processor using the NTA reference.
PREFETCHT0 <i>mem8</i>	0F 18 /1	Move data closer to the processor using the T0 reference.
PREFETCHT1 <i>mem8</i>	0F 18 /2	Move data closer to the processor using the T1 reference.
PREFETCHT2 <i>mem8</i>	0F 18 /3	Move data closer to the processor using the T2 reference.

Table 2-4. Locality References for the Prefetch Instructions

Locality Reference	Description
NTA	Non-Temporal Access—Move the specified data into the processor with minimum cache pollution. This is intended for data that will be used accessed only once, rather than repeatedly. The specific technique for minimizing cache pollution is implementation-dependent and may include such techniques as allocating space in a software-invisible buffer, allocating a cache line in only a single way, etc. For details, see the software-optimization documentation for a particular hardware implementation.
T0	All Cache Levels—Move the specified data into all cache levels.
T1	Level 2 and Higher—Move the specified data into all cache levels except 0th level (L1) cache.
T2	Level 3 and Higher—Move the specified data into all cache levels except 0th level (L1) and 1st level (L2) caches.

The operation of this instruction is implementation-dependent. The instructions can be ignored or changed by a processor implementation. The size of the cache line also depends on the implementation, with a minimum size of 32 bytes. For details on the use of this instruction, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

Related Instructions

PREFETCH, PREFETCHW

rFLAGS Affected

None.

Exceptions

None.

PUSH**Push onto Stack**

Decrements the stack pointer and stores the source operand on the top of the stack. The address-size attribute of the stack segment determines the stack pointer size (16, 32, or 64 bits), and the operand-size attribute of the current code segment determines the amount the stack pointer is decremented (2, 4, or 8 according to the number of bytes). Pushing an odd number of 16-bit operands when the stack address-size attribute is 32 results in a misaligned the stack pointer.

The PUSH rSP instruction pushes the value of the rSP register as it existed before the instruction was executed.

There is a PUSH CS instruction, but there is no corresponding POP CS. A value from the top of stack is put into the CS register while performing the RET instruction.

In 64-bit mode, the operand size of all PUSH instruction defaults to 64 bits, and there is no prefix available for encoding a 32-bit operand size. An invalid-opcode exception occurs if the 0E, 16, 1E, or 06 opcode of this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
PUSH <i>reg/mem16</i>	FF /6	Pushes the contents of a 16-bit register or memory operand onto the stack and decrements the stack pointer.
PUSH <i>reg/mem32</i>	FF /6	Pushes the contents of a 32-bit register or memory operand onto the stack and decrements the stack pointer.
PUSH <i>reg/mem64</i>	FF /6	Pushes the contents of a 64-bit register or memory operand onto the stack and decrements the stack pointer.
PUSH <i>reg16</i>	50 + <i>rw</i>	Pushes the contents of a 16-bit register onto the stack and decrements the stack pointer.
PUSH <i>reg32</i>	50 + <i>rd</i>	Pushes the contents of a 32-bit register onto the stack and decrements the stack pointer.
PUSH <i>reg64</i>	50 + <i>rq</i>	Pushes the contents of a 64-bit register onto the stack and decrements the stack pointer.
PUSH <i>imm8</i>	6A	Push an 8-bit immediate value onto the stack and decrement the stack pointer.
PUSH <i>imm16</i>	68	Push a 16-bit immediate value onto the stack and decrement the stack pointer.
PUSH <i>imm32</i>	68	Push a 32-bit immediate value onto the stack and decrement the stack pointer.
PUSH CS	0E	Push the contents of the CS register onto the stack and decrement the stack pointer. (Invalid in 64-bit mode.)
PUSH SS	16	Push the contents of the SS register onto the stack and decrement the stack pointer. (Invalid in 64-bit mode.)

Mnemonic	Opcode	Description
PUSH DS	1E	Push the contents of the DS register onto the stack and decrement the stack pointer. (Invalid in 64-bit mode.)
PUSH ES	06	Push the contents of the DS register onto the stack and decrement the stack pointer. (Invalid in 64-bit mode.)
PUSH FS	0F A0	Push the contents of the DS register onto the stack and decrement the stack pointer.
PUSH GS	0F A8	Push the contents of the GS register onto the stack and decrement the stack pointer.

Related Instructions

PUSHA, PUSHAD, PUSHAX, PUSHF, PUSHFD, PUSHFQ, POP, POPA, POPAD, POPF, POPFD, PUSHFQ.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	The 0E, 1E, 06, and 16 opcodes of this instruction are not recognized in 64-bit mode.
Stack, #SS	X X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.) The new value of the SP or ESP register exceeded the stack segment limit.
General protection, #GP	X	X	X X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

Push All GPRs onto Stack

Push the contents of the general-purpose registers eAX, eCX, eDX, eBX, eSP, (original value), eBP, eSI, and eDI onto the stack in the ordered specified.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
PUSHA	60	Push the contents of the AX, CX, DX, BX, original SP, BP, SI, and DI onto the stack. (Invalid in 64-bit mode.)
PUSHAD	60	Push the contents of the EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI onto the stack. (Invalid in 64-bit mode.)

Related Instructions

PUSH, PUSHF, PUSHFD, PUSHFQ, POP, POPA, POPAD, POPF, POPFD, PUSHFQ.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.
Stack, #SS	X	X	X	The starting or ending stack address exceeded the stack segment limit. There is not enough stack space.
General protection, segment overrun, #GP	X	X		The ESP or SP register contains 7, 9, 11, 13, or 15.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

PUSHF

PUSHFD

PUSHFQ

Push rFLAGS Word onto Stack

Copies the bits of the rFLAGS register onto the stack and decrements rSP by 2, 4 or 8.

When copying the entire rFLAGS register to the stack, the VM and RF flags (bits 16 and 17) are not copied; instead, the values for these flags are cleared in the rFLAGS image stored on the stack.

When operating in virtual-8086 mode, if system software has set the IOPL field to a value less than 3, a general-protection exception occurs if application software attempts to execute a POPF, POPFD, POPFQ, PUSHF, PUSHFD, or PUSHFQ instruction. If system software has set the IOPL field to a value of 3, application software can cause the interrupt flag, rFLAGS.IF (bit 9) to be cleared to 0 by executing a CLI instruction. For a description on how this is done, and for information about how system software accesses the system bits in rFLAGS (those bits that can be accessed only by system software), see Section 8, “Exceptions and Interrupts”, on page 253 of Volume 2.

Mnemonic	Opcode	Description
PUSHF	9C	Push the flags register onto the stack.
PUSHFD	9C	Push eFLAGS doubleword onto stack.
PUSHFQ	9C	Push rFLAGS doubleword onto stack.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

PUSHF_START:

```
IF (REAL_MODE)
    PUSHF_REAL
ELSIF (PROTECTED_MODE)
    PUSHF_PROTECTED
ELSE // (VIRTUAL_MODE)
    PUSHF_VIRTUAL
```

PUSHF_REAL:

```
PUSH.v old_rFLAGS    // pushed with RF clear
EXIT
```

PUSHF_PROTECTED:

```
PUSH.v old_RFLAGS    // pushed with RF clear
EXIT
```

PUSHF_VIRTUAL:

```
IF (RFLAGS.IOPL=3)
{
    PUSH.v old_RFLAGS // pushed with RF,VM clear
    EXIT
}
ELIF ((CR4.VME=1) && (OPERAND_SIZE=16))
{
    PUSH.v old_RFLAGS // pushed with VIF in the IF position
                      // pushed with IOPL=3
    EXIT
}
ELSE // ((RFLAGS.IOPL<3) && ((CR4.VME=0) || (OPERAND_SIZE!=16)))
    EXCEPTION [#GP(0)]
```

Related Instructions

PUSHA, PUSHAD, PUSHAX, PUSHF, PUSHFD, PUSHFQ, POP, POPA, POPAD, POPF, POPFD.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS			X	The new value of the ESP register exceeded the stack segment limit.
General protection, #GP		X		The I/O privilege level is less than 3.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

RCL Rotate Through Carry Left

Rotates the bits of the first operand to the left and through the carry flag by the number of bit positions specified in the second operand. The bits rotated through the carry flag are rotated back in at the left end (MSB).

The affected operand can be a register or a memory location; the count operand is an unsigned integer that can be an immediate or a value in the CL register. The processor restricts the count to a number between 0 and 31 by masking all the bits in the count operand except the 5 least-significant bits. In 64-bit mode, the count mask is 6 bits wide which provides a range of 0 to 63.

For 1-bit rotates, the OF flag is set to the exclusive OR of the CF bit (after the rotate) and the most significant bit of the result.

Mnemonic	Opcode	Description
<i>RCL reg/mem8, 1</i>	<i>D0 /2</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand left once.
<i>RCL reg/mem8, CL</i>	<i>D2 /2</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand left the number of bits specified in the CL register.
<i>RCL reg/mem8, imm8</i>	<i>C0 /2 ib</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand left the number of bits specified by an 8-bit immediate value.
<i>RCL reg/mem16, 1</i>	<i>D1 /2</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand left once.
<i>RCL reg/mem16, CL</i>	<i>D3 /2</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand left the number of bits specified in the CL register.
<i>RCL reg/mem16, imm8</i>	<i>C1 /2 ib</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand left the number of bits specified by an 8-bit immediate value.
<i>RCL reg/mem32, 1</i>	<i>D1 /2</i>	Rotate the 33 bits consisting of the carry flag and a 32-bit register or memory operand left once.
<i>RCL reg/mem32, CL</i>	<i>D3 /2</i>	Rotate 33 bits consisting of the carry flag and a 32-bit register or memory operand left the number of bits specified in the CL register.
<i>RCL reg/mem32, imm8</i>	<i>C1 /2 ib</i>	Rotate the 33 bits consisting of the carry flag and a 32-bit register or memory operand left the number of bits specified by an 8-bit immediate value.

RCL <i>reg/mem64</i> , 1	D1 /2	Rotate the 65 bits consisting of the carry flag and a 64-bit register or memory operand left once.
RCL <i>reg/mem64</i> , CL	D3 /2	Rotate 65 bits consisting of the carry flag and a 64-bit register or memory operand left the number of bits specified in the CL register.
RCL <i>reg/mem64</i> , <i>imm8</i>	C1 /2 <i>ib</i>	Rotate the 65 bits consisting of the carry flag and a 64-bit register or memory operand left the number of bits specified by an 8-bit immediate value.

Related Instructions

RCR, ROL, ROR

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0								1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:

Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The source operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

RCR Rotate Through Carry Right

Rotates the bits of the first operand to the right and through the carry flag by the number of bit positions specified in the second operand. The bits rotated through the carry flag are rotated back in at the left end (MSB).

The affected operand can be a register or a memory location; the count operand is an unsigned integer that can be an immediate or a value in the CL register. The processor restricts the count to a number between 0 and 31 by masking all the bits in the count operand except the 5 least-significant bits. In 64-bit mode, the count mask is 6 bits wide which provides a range of 0 to 63.

For 1-bit rotates, the OF flag is set to the exclusive OR of the two most significant bits of the result.

Mnemonic	Opcode	Description
<i>RCR reg/mem8, 1</i>	<i>D0 /3</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand right once.
<i>RCR reg/mem8, CL</i>	<i>D2 /3</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand right the number of bits specified in the CL register.
<i>RCR reg/mem8, imm8</i>	<i>C0 /3 ib</i>	Rotate the 9 bits consisting of the carry flag and an 8-bit register or memory operand right the number of bits specified by an 8-bit immediate value.
<i>RCR reg/mem16, 1</i>	<i>D1 /3</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand right once.
<i>RCR reg/mem16, CL</i>	<i>D3 /3</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand right the number of bits specified in the CL register.
<i>RCR reg/mem16, imm8</i>	<i>C1 /3 ib</i>	Rotate the 17 bits consisting of the carry flag and a 16-bit register or memory operand right the number of bits specified by an 8-bit immediate value.
<i>RCR reg/mem32, 1</i>	<i>D1 /3</i>	Rotate the 33 bits consisting of the carry flag and a 32-bit register or memory operand right once.
<i>RCR reg/mem32, CL</i>	<i>D3 /3</i>	Rotate 33 bits consisting of the carry flag and a 32-bit register or memory operand right the number of bits specified in the CL register.
<i>RCR reg/mem32, imm8</i>	<i>C1 /3 ib</i>	Rotate the 33 bits consisting of the carry flag and a 32-bit register or memory operand right the number of bits specified by an 8-bit immediate value.

RCR reg/mem64,1	D1 /3	Rotate the 65 bits consisting of the carry flag and a 64-bit register or memory operand right once.
RCR reg/mem64,CL	D3 /3	Rotate 65 bits consisting of the carry flag and a 64-bit register or memory operand right the number of bits specified in the CL register.
RCR reg/mem64, imm8	C1 /3 ib	Rotate the 65 bits consisting of the carry flag and a 64-bit register or memory operand right the number of bits specified by an 8-bit immediate value.

Related Instructions

RCL, ROR, ROL

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0								1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The source operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

RET**Return from Called Procedure**

Used to return from a procedure previously entered by a CALL instruction. The IP is restored from the stack and the SP is incremented by 2. If the procedure was FAR, then RET (return FAR) is used, and in addition to restoring the IP, the CS is restored from the stack and SP is again incremented by 2. The RET instruction may be followed by a number that will be added to the SP after the SP has been incremented. This is done to skip over any parameters being passed back to the calling program segment. The RET instruction can be used to execute three different types of returns:

- Near return—A return to a calling procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment return.
- Far return—A return to a calling procedure located in a different segment than the current code segment, sometimes referred to as an intersegment return.
- Interprivilege-level far return—A far return to a different privilege level than that of the currently executing program or procedure.

In 64-bit mode, for near returns only (not far returns), the operand size defaults to 64 bits without the need for a REX prefix.

Mnemonic	Opcode	Description
RET	C3	Near return to the calling procedure.
RET	CB	Far return to the calling procedure.
RET <i>imm16</i>	C2 <i>iw</i>	Near return to the calling procedure, with pop of the number of bytes specified by an immediate word operand from the stack.
RET <i>imm16</i>	CA <i>iw</i>	Far return to the calling procedure, with pop of the number of bytes specified by an immediate word operand from the stack.

For details about control-flow instructions, see Section 3.7, “Control Transfers”, on page 95 of Volume 1, and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.

Action

```
// Far returns (RETF)
// See Section 2.5.3, “Pseudocode Definitions”, on page 46.
```

```
RETF_START:
```

```
IF (REAL_MODE)
    RETF_REAL_OR_VIRTUAL
```

```

ELSIF (PROTECTED_MODE)
    RETF_PROTECTED
ELSE // (VIRTUAL_MODE)
    RETF_REAL_OR_VIRTUAL

```

RETF_REAL_OR_VIRTUAL:

```

    IF (OPCODE = retf imm16)
        temp_IMM = word-sized immediate specified in the instruction,
                    zero-extended to 64 bits
    ELSE // (OPCODE = retf)
        temp_IMM = 0

    POP.v temp_RIP
    POP.v temp_CS

    IF (temp_RIP > CS.limit)
        EXCEPTION [#GP(0)]

    CS.sel = temp_CS
    CS.base = temp_CS SHL 4

    RSP.s = RSP + temp_IMM
    RIP = temp_RIP
    EXIT

```

RETF_PROTECTED:

```

    IF (OPCODE = retf imm16)
        temp_IMM = word-sized immediate specified in the instruction,
                    zero-extended to 64 bits
    ELSE // (OPCODE = retf)
        temp_IMM = 0

    POP.v temp_RIP
    POP.v temp_CS

    temp_CPL = temp_CS.rpl

    IF (CPL=temp_CPL)
    {
        CS = READ_DESCRIPTOR (temp_CS, iret_chk)

        RSP.s = RSP + temp_IMM

        IF ((64BIT_MODE) && (temp_RIP is non-canonical)
            || (!64BIT_MODE) && (temp_RIP > CS.limit))
            EXCEPTION [#GP(0)]
    }

```



```

        RIP = temp_RIP
        EXIT
    }
ELSE // (CPL!=temp_CPL)
{
    RSP.s = RSP + temp_IMM

    POP.v temp_RSP
    POP.v temp_SS

    CS = READ_DESCRIPTOR (temp_CS, iret_chk)

    CPL = temp_CPL

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
        EXCEPTION [#GP(0)]

    SS = READ_DESCRIPTOR (temp_SS, ss_chk)

    IF ((DS.attr.dpl < CPL)
        && ((DS.attr.type = 'data') || (DS.attr.type = 'non-conforming-code'))))
        DS = NULL // can't use lower dpl data segment at higher cpl

    IF ((ES.attr.dpl < CPL)
        && ((ES.attr.type = 'data') || (ES.attr.type = 'non-conforming-code'))))
        ES = NULL // can't use lower dpl data segment at higher cpl

    IF ((FS.attr.dpl < CPL)
        && ((FS.attr.type = 'data') || (FS.attr.type = 'non-conforming-code'))))
        FS = NULL // can't use lower dpl data segment at higher cpl

    IF ((GS.attr.dpl < CPL)
        && ((GS.attr.type = 'data') || (GS.attr.type = 'non-conforming-code'))))
        GS = NULL // can't use lower dpl data segment at higher cpl

    RSP.s = temp_RSP
    RIP = temp_RIP
    EXIT
}

```

Related Instructions

CALL

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Segment not present, #NP (selector)			X	The return code or stack segment is not present.
Stack, #SS	X	X	X	The top bytes of the stack are not within stack limits.
			X	The return stack segment is not present.
General protection, #GP			X	The return instruction pointer is not within the return code segment limit.
			X	The return code or stack segment selector is null.
General protection, #GP (selector)			X	The return code or the stack segment selector index is outside the descriptor table limits.
			X	The RPL of the return code segment selector is less than the CPL.
			X	The return code segment is nonconforming and the segment selector's DPL is not equal to the RPL of the code segment's segment selector.
			X	The return code segment is conforming and the segment selector's DPL is greater than the RPL of the code segment's segment selector
			X	The stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
			X	The stack segment is not a writable data segment.
			X	The stack segment selector RPL is not equal to the RPL of the return code segment selector.
General protection, segment overrun, #GP	X	X		The segment descriptor for the return code does not indicate that it is a code segment.
				The return address pointer falls outside the address range of 00000h to FFFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

ROL Rotate Left

Rotates the bits of the first operand to the right the number of bit positions specified in the second operand. The bits rotated out left are rotated back in at the right end (LSB).

The affected operand can be a register or a memory location; the count operand is an unsigned integer that can be an immediate or a value in the CL register.

When the rotate count is 1, the overflow flag (OF) is set to the XOR of the two most-significant bits. The value of the overflow flag is undefined for multi-bit rotates. The value of the carry bit is set to the value of the bit rotated out from the left.

Mnemonic	Opcode	Description
ROL <i>reg/mem8</i> , 1	D0 /0	Rotate an 8-bit register or memory operand left once.
ROL <i>reg/mem8</i> , CL	D2 /0	Rotate an 8-bit register or memory operand left the number of bits specified in the CL register.
ROL <i>reg/mem8</i> , <i>imm8</i>	C0 /0 <i>ib</i>	Rotate an 8-bit register or memory operand left the number of bits specified by an 8-bit immediate value.
ROL <i>reg/mem16</i> , 1	D1 /0	Rotate a 16-bit register or memory operand left once.
ROL <i>reg/mem16</i> , CL	D3 /0	Rotate a 16-bit register or memory operand left the number of bits specified in the CL register.
ROL <i>reg/mem16</i> , <i>imm8</i>	C1 /0 <i>ib</i>	Rotate a 16-bit register or memory operand left the number of bits specified by an 8-bit immediate value.
ROL <i>reg/mem32</i> , 1	D1 /0	Rotate a 32-bit register or memory operand left once.
ROL <i>reg/mem32</i> , CL	D3 /0	Rotate a 16-bit register or memory operand left the number of bits specified in the CL register.
ROL <i>reg/mem32</i> , <i>imm8</i>	C1 /0 <i>ib</i>	Rotate a 16-bit register or memory operand left the number of bits specified by an 8-bit immediate value.
ROL <i>reg/mem64</i> , 1	D1 /0	Rotate a 64-bit register or memory operand left once.
ROL <i>reg/mem64</i> , CL	D3 /0	Rotate a 64-bit register or memory operand left the number of bits specified in the CL register.
ROL <i>reg/mem64</i> , <i>imm8</i>	C1 /0 <i>ib</i>	Rotate a 64-bit register or memory operand left the number of bits specified by an 8-bit immediate value.

Related Instructions

RCL, RCR, ROR, SHR, SHL

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0								1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The source operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

ROR Rotate Right

Rotates the bits of the first operand to the right the number of bit positions specified in the second operand. The bits rotated out right are rotated back in at the left end (MSB).

The affected operand can be a register or a memory location; the count operand is an unsigned integer that can be an immediate or a value in the CL register. The processor restricts the count to a number between 0 and 31 by masking all the bits in the count operand except the 5 least-significant bits. In 64-bit mode, the count mask is 6 bits wide which provides a range of 0 to 63.

In single-bit rotates, the overflow flag (OF) is set to the XOR of the two most-significant bits. The value of the overflow flag is undefined for multi-bit rotates. The value of the carry bit is set to the value of the bit rotated out from the right.

Mnemonic	Opcode	Description
ROR <i>reg/mem8</i> , 1	D0 /1	Rotate an 8-bit register or memory operand right once.
ROR <i>reg/mem8</i> , CL	D2 /1	Rotate an 8-bit register or memory operand right the number of bits specified in the CL register.
ROR <i>reg/mem8</i> , <i>imm8</i>	C0 /1 <i>ib</i>	Rotate an 8-bit register or memory operand right the number of bits specified by an 8-bit immediate value.
ROR <i>reg/mem16</i> , 1	D1 /1	Rotate a 16-bit register or memory operand right once.
ROR <i>reg/mem16</i> , CL	D3 /1	Rotate a 16-bit register or memory operand right the number of bits specified in the CL register.
ROR <i>reg/mem16</i> , <i>imm8</i>	C1 /1 <i>ib</i>	Rotate a 16-bit register or memory operand right the number of bits specified by an 8-bit immediate value.
ROR <i>reg/mem32</i> , 1	D1 /1	Rotate a 32-bit register or memory operand right once.
ROR <i>reg/mem32</i> , CL	D3 /13	Rotate a 32-bit register or memory operand right the number of bits specified in the CL register.
ROR <i>reg/mem32</i> , <i>imm8</i>	C1 /1 <i>ib</i>	Rotate a 32-bit register or memory operand right the number of bits specified by an 8-bit immediate value.
ROR <i>reg/mem64</i> , 1	D1 /1	Rotate a 64-bit register or memory operand right once.
ROR <i>reg/mem64</i> , CL	D3 /1	Rotate a 64-bit register or memory operand right the number of bits specified in the CL register.
ROR <i>reg/mem64</i> , <i>imm8</i>	C1 /1 <i>ib</i>	Rotate a 64-bit register or memory operand right the number of bits specified by an 8-bit immediate value.

Related Instructions

RCL, RCR, ROL, SHR, SHL

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0								1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:

Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The source operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SAHF Store AH into Flags

Loads the SF, ZF, AF, PF, and CF flags of the EFLAGS register with values from the corresponding bits in the AH register (bits 7, 6, 4, 2, and 0, respectively). Bits 1, 3, and 5 of register AH are ignored; the corresponding reserved bits (1, 3, and 5) in the EFLAGS register remain as shown in the “Operation” section below.

An invalid-opcode exception occurs if this instruction is used in 64-bit mode.

Mnemonic	Opcode	Description
SAHF	9E	Loads the sign flag, the zero flag, the auxiliary flag, the parity flag, and the carry flag from the AH register into the lower 8 bits of the EFLAGS register. (Invalid in 64-bit mode.)

Related Instructions

LAHF

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
												1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X	Software is executing in 64-bit mode.

SAL**Shift Arithmetic Left**

Shifts the bits in the first operand (destination operand) to the left by the number of bits specified in the second operand (count operand). Bits shifted beyond the destination operand boundary are first shifted into the CF flag, then discarded. At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.

The destination operand can be a register or a memory location. The count operand can be an immediate value or register CL. The count is masked to 5 bits, which limits the count range to 0 to 31. In 64-bit mode, the count mask is 6 bits wide which provides a range of 0 to 63. A special opcode encoding is provided for a count of 1.

This instruction is synonymous with the SHL instruction.

Mnemonic	Opcode	Description
<i>SAL reg/mem8, 1</i>	D0 /4	Multiply an 8-bit register or memory operand by two.
<i>SAL reg/mem8, CL</i>	D2 /4	Multiply an 8-bit register or memory operand by two the number of times specified in the CL register.
<i>SAL reg/mem8, imm8</i>	C0 /4 <i>ib</i>	Multiply an 8-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
<i>SAL reg/mem16, 1</i>	D1 /4	Multiply a 16-bit register or memory operand by two.
<i>SAL reg/mem16, CL</i>	D3 /4	Multiply a 16-bit register or memory operand by two the number of times specified in the CL register.
<i>SAL reg/mem16, imm8</i>	C1 /4 <i>ib</i>	Multiply a 16-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
<i>SAL reg/mem32, 1</i>	D1 /4	Multiply a 32-bit register or memory operand by two.
<i>SAL reg/mem32, CL</i>	D3 /4	Multiply a 32-bit register or memory operand by two the number of times specified in the CL register.
<i>SAL reg/mem32, imm8</i>	C1 /4 <i>ib</i>	Multiply a 32-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
<i>SAL reg/mem64, 1</i>	D1 /4	Multiply a 64-bit register or memory operand by two.
<i>SAL reg/mem64, CL</i>	D3 /4	Multiply a 64-bit register or memory operand by two the number of times specified in the CL register.
<i>SAL reg/mem64, imm8</i>	C1 /4 <i>ib</i>	Multiply a 64-bit register or memory operand by two the number of times specified by an 8-bit immediate value.

Related Instructions

SAR, SHL, SHR, SHLD, SHRD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SAR**Shift Arithmetic Right**

Shifts the bits in the first operand (destination operand) to the right by the number of bits specified in the second operand (count operand). Bits shifted beyond the destination operand boundary are first shifted into the CF flag, then discarded. At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.

The destination operand can be a register or a memory location. The count operand can be an immediate value or register CL. The count is masked to 5 bits, which limits the count range to 0 to 31. In 64-bit mode, the count mask is 6 bits wide which provides a range of 0 to 63. A special opcode encoding is provided for a count of 1.

The SAR instruction treats the most-significant bit of an operand in a special way: this bit (the sign bit) is not changed, but is copied to the next bit, preserving the sign of the result. The least-significant bit is shifted out to the CF flag. In the SAL instruction, the most-significant bit is shifted out to CF flag, and the least-significant bit is cleared to 0.

Although the SAR instruction divides the operand by a power of 2, the behavior is different from the IDIV instruction. For example, shifting -11 (FFFFFFF5h) by two bits to the right (i.e. divide -11 by 4), gives a result of FFFFFFFDh, or -3, whereas the IDIV instruction for dividing -11 by 4 gives a result of -2. This is because the IDIV instruction rounds off the quotient to zero, whereas the SAR instruction rounds off the remainder to zero for positive dividends, and to negative infinity for negative dividends. This means that, for positive operands, SAR behaves like the corresponding IDIV instruction, and for negative operands, it gives the same result if and only if all the shifted-out bits are zeroes, and otherwise the result is smaller by 1.

Mnemonic	Opcode	Description
<i>SAR reg/mem8, 1</i>	D0 /7	Signed divided an 8-bit register or memory operand by two.
<i>SAR reg/mem8, CL</i>	D2 /7	Signed divide an 8-bit register or memory operand by two the number of times specified in the CL register.
<i>SAR reg/mem8, imm8</i>	C0 /7 ib	Signed divide an 8-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
<i>SAR reg/mem16, 1</i>	D1 /7	Signed divide a 16-bit register or memory operand by two.
<i>SAR reg/mem16, CL</i>	D3 /7	Signed divide a 16-bit register or memory operand by two the number of times specified in the CL register.
<i>SAR reg/mem16, imm8</i>	C1 /7 ib	Signed divide a 16-bit register or memory operand by two the number of times specified by an 8-bit immediate value.

SAR reg/mem32, 1	D1 /7	Signed divide a 32-bit register or memory operand by two.
SAR reg/mem32, CL	D3 /7	Signed divide a 32-bit register or memory operand by two the number of times specified in the CL register.
SAR reg/mem32, imm8	C1 /7 ib	Signed divide a 32-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SAR reg/mem64, 1	D1 /7	Signed divide a 64-bit register or memory operand by two.
SAR reg/mem64, CL	D3 /7	Signed divide a 64-bit register or memory operand by two the number of times specified in the CL register.
SAR reg/mem64, imm8	C1 /7 ib	Signed divide a 64-bit register or memory operand by two the number of times specified by an 8-bit immediate value.

Related Instructions

SAL, SHL, SHR, SHLD, SHRD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SBB**Subtract with Borrow**

Subtracts source operand from destination, replacing destination. If the carry flag is 1, SBB subtracts 1 from the result. Otherwise, it executes like SUB. This instruction is useful for multibyte (multiword) numbers. It takes care of the borrow of the lower operand.

The destination operand can be a register or a memory location; the source operand can be an immediate value, a register, or a memory location.

Two memory operands cannot be used in one instruction. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The LOCK prefix can be used with forms of the SBB instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
SBB AL, <i>imm8</i>	1C <i>ib</i>	Subtract an immediate 8-bit value from the AL register with borrow.
SBB AX, <i>imm16</i>	1D <i>iw</i>	Subtract an immediate 16-bit value from the AX register with borrow.
SBB EAX, <i>imm32</i>	1D <i>id</i>	Subtract an immediate 32-bit value from the EAX register with borrow.
SBB RAX, <i>imm32</i>	1D <i>id</i>	Subtract an immediate 32-bit value from the RAX register with borrow.
SBB <i>reg/mem8</i> , <i>imm8</i>	80 /3 <i>ib</i>	Subtract an immediate 8-bit value from an 8-bit register or memory operand with borrow.
SBB <i>reg/mem16</i> , <i>imm16</i>	81 /3 <i>iw</i>	Subtract an immediate 16-bit value from a 16-bit register or memory operand with borrow.
SBB <i>reg/mem32</i> , <i>imm32</i>	81 /3 <i>id</i>	Subtract an immediate 32-bit value from a 32-bit register or memory operand with borrow.
SBB <i>reg/mem64</i> , <i>imm32</i>	81 /3 <i>id</i>	Subtract an immediate 32-bit value from a 64-bit register or memory operand with borrow.
SBB <i>reg/mem16</i> , <i>imm8</i>	83 /3 <i>ib</i>	Subtract a sign-extended 8-bit immediate value from a 16-bit register or memory operand.
SBB <i>reg/mem32</i> , <i>imm8</i>	83 /3 <i>ib</i>	Subtract a sign-extended 8-bit immediate value from a 32-bit register or memory operand.

Mnemonic	Opcode	Description
<i>SBB reg/mem64, imm8</i>	<i>83 /3 ib</i>	Subtract a sign-extended 8-bit immediate value from a 64-bit register or memory operand.
<i>SBB reg/mem8, reg8</i>	<i>18 /r</i>	Subtract the contents of an 8-bit register from an 8-bit register or memory operand.
<i>SBB reg/mem16, reg16</i>	<i>19 /r</i>	Subtract the contents of an 16-bit register from a 16-bit register or memory operand.
<i>SBB reg/mem32, reg32</i>	<i>19 /r</i>	Subtract the contents of an 32-bit register from a 32-bit register or memory operand.
<i>SBB reg/mem64, reg64</i>	<i>19 /r</i>	Subtract the contents of an 64-bit register from a 64-bit register or memory operand.
<i>SBB reg8, reg/mem8</i>	<i>1A /r</i>	Subtract the contents of an 8-bit register or memory operand from the contents of an 8-bit register
<i>SBB reg16, reg/mem16</i>	<i>1B /r</i>	Subtract the contents of an 16-bit register or memory operand from the contents of an 16-bit register
<i>SBB reg32, reg/mem32</i>	<i>1B /r</i>	Subtract the contents of an 32-bit register or memory operand from the contents of an 32-bit register
<i>SBB reg64, reg/mem64</i>	<i>1B /r</i>	Subtract the contents of an 64-bit register or memory operand from the contents of an 64-bit register

Related Instructions

SUB, FSUB, SISUB

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SCAS Scan String

SCASB

SCASW

SCASD

SCASQ

Compares the byte, word, doubleword, or quadword specified with the memory operand with the value in the AL, AX, EAX, or RAX register, and sets flags in the rFLAGS register according to the results. The memory operand address is read from the DI, EDI, or RDI register, depending upon the operand size.

After the comparison, the DI register is incremented or decremented automatically according to the setting of the DF flag in the rFLAGS register. If the DF flag is 0, the DI register is incremented by the number of bytes in the operand; if the DF flag is 1, the DI register is decremented by the number of bytes in the operand.

The forms of the SCASx instruction that have an explicit operand are equivalent to the forms with implicit operand (in fact, they have the same opcodes). The explicit forms, however, only allow specification of the operand size. The address of the explicit operand is otherwise ignored, and instead, the implicit operand—[rDI]—is always used.

The REPE or REPZ prefixes (they are synonyms) and the REPNE or REPNZ prefixes (they are synonyms) can be used with the SCASx instructions. For details about the REP prefix, see “Repeat Prefixes” on page 10. The SCASx instructions can also be put inside a loop controlled by the LOOPcc instruction.

Mnemonic	Opcode	Description
SCAS mem8	AE	Compare the contents of the AL register with the byte at DI and set status flags to reflect the outcome of the operation.
SCAS mem16	AF	Compare the contents of the AX register with the word at DI and set status flags to reflect the outcome of the operation.
SCAS mem32	AF	Compare the contents of the EAX register with the doubleword at EDI and set status flags to reflect the outcome of the operation.
SCAS mem64	AF	Compare the contents of the RAX register with the doubleword at RDI and set status flags to reflect the outcome of the operation.

SCASB	AE	Compare the contents of the AL register with the byte at DI and set status flags to reflect the outcome of the operation.
SCASW	AF	Compare the contents of the AX register with the word at DI and set status flags to reflect the outcome of the operation.
SCASD	AF	Compare the contents of the RAX register with the doubleword at RDI and set status flags to reflect the outcome of the operation.
SCASQ	AF	Compare the contents of the RAX register with the quadword at RDI and set status flags to reflect the outcome of the operation.

Related Instructions

SCASB, SCASD, SCASQ, SCASW

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X		During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of the ES segment used for the operand, points to an illegal memory location.
			X	The ES register contains a null segment selector.
			X	The memory operand effective address in the ES segment is illegal.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SETcc Set Byte if Above

The SETcc instructions set an 8-bit memory or register value to 1 depending on the settings in rFLAGS.

Mnemonics with the A (above) and (B) tags are intended for use when performing unsigned integer comparisons; Those with G (greater) and L (lesser) tags are intended for use with signed integer comparisons.

Mnemonic	Opcode	Description
SETA <i>reg/mem8</i> SETNB <i>reg/mem8</i>	0F 97	Set byte if the carry flag and the zero flag are both zero.
SETAE <i>reg/mem8</i> SETNB <i>reg/mem8</i> SETNC <i>reg/mem8</i>	0F 93	Set byte if the carry flag is zero.
SETB <i>reg/mem8</i> SETC <i>reg/mem8</i> SETNAE <i>reg/mem8</i>	0F 92	Set byte if the carry flag is 1.
SETBE <i>reg/mem8</i> SETNA <i>reg/mem8</i>	0F 96	Set byte if the carry flag is 1 or the zero flag is 1.
SETE <i>reg/mem8</i> SETZ <i>reg/mem8</i>	0F 94	Set byte if the zero flag is 1.
SETG <i>reg/mem8</i> SETNLE <i>reg/mem8</i>	0F 9F	Set byte if the zero flag and the sign flag is equal to the overflow flag.
SETGE <i>reg/mem8</i> SETNL <i>reg/mem8</i>	0F 9D	Set byte if the sign flag is equal to the overflow flag.
SETL <i>reg/mem8</i> SETNGE <i>reg/mem8</i>	0F 9C	Set byte if the sign flag is not equal to the overflow flag.
SETLE <i>reg/mem8</i> SETNG <i>reg/mem8</i>	0F 9E	Set byte if the zero flag is 1 or the sign flag and the overflow flag are not equal.
SETNE <i>reg/mem8</i> SETNZ <i>reg/mem8</i>	0F 95	Set byte if the zero flag is zero.
SETNO <i>reg/mem8</i>	0F 91	Set byte if the overflow flag is zero.
SETNP <i>reg/mem8</i> SETPO <i>reg/mem8</i>	0F 9B	Set byte if the parity flag is zero.
SETNS <i>reg/mem8</i>	0F 99	Set byte if the sign flag is zero.
SETO <i>reg/mem8</i>	0F 90	Set byte if the overflow flag is 1.
SETP <i>reg/mem8</i> SETPE <i>reg/mem8</i>	0F 9A	Set byte if the parity flag is 1.

SETS *reg/mem8*

OF 98

Set byte if the sign flag is 1.

SETcc instructions are often used to set logical indicators. Like CMOVcc instructions (page 102), SETcc instructions can replace two instructions—a conditional jump and a move. Replacing conditional jumps with conditional sets can help avoid branch-prediction penalties that may be caused by conditional jumps.

Some languages represent a logical 1 as an integer with all bits set. This representation can be obtained by choosing the logically opposite condition for the instruction, the decrementing by 1.

If the logical value “true” (logical one) is represented in a high-level language as an integer with all bits set to 1, software can accomplish such representation by first executing the opposite SETcc instruction—for example, the opposite of SETZ is SETNZ—and then decrementing the result.

Related Instructions

None.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X		An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SFENCE Store Fence

The SFENCE instruction acts as a barrier to force strong memory ordering (serialization) between store instructions preceding the SFENCE and store instructions that follow the SFENCE. In a weakly-ordered memory system, hardware is allowed to reorder reads and writes between the processor and memory. The SFENCE instruction guarantees that all prior stores complete before subsequent stores are executed.

SFENCE is weakly-ordered with respect to load instructions, data and instruction prefetches, and the SFENCE instruction. Speculative loads initiated by the processor, or specified explicitly using cache-prefetch instructions, can be reordered around an SFENCE.

In addition to store instructions, SFENCE is strongly ordered with respect to other SFENCE instructions, MFENCE instructions, and serializing instructions.

Support for the SFENCE instruction is indicated when the SSE bit (bit 25) is set to 1 in EDX after executing CUID with extended function 8000_0001h.

Mnemonic	Opcode	Description
SFENCE	0F AE F8	Force strong ordering of (serialize) store operations.

Related Instructions

LFENCE, MFENCE

rFLAGS Affected

None.

Exceptions

None.

SHL Shift Left

This instruction shifts bits in a destination register or memory operand to the left by the number of bits specified by a source operand. On each bit shift, the most-significant bit is shifted into the carry flag and the least-significant bit is cleared to zero. This instruction is the same as the SAL instruction. This instruction is used to perform multiplication by powers of two.

The overflow flag (OF) is set equal to the most significant bit of the original operand on 1-bit shifts. The SF, ZF, and PF flags are set according to the result. If the shift count is 0, no flags are affected.

Mnemonic	Opcode	Description
SHL <i>reg/mem8</i> , 1	D0 /4	Multiply an 8-bit register or memory operand by two.
SHL <i>reg/mem8</i> , CL	D2 /4	Multiply an 8-bit register or memory operand by two the number of times specified in the CL register.
SHL <i>reg/mem8</i> , <i>imm8</i>	C0 /4 <i>ib</i>	Multiply an 8-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHL <i>reg/mem16</i> , 1	D1 /4	Multiply a 16-bit register or memory operand by two.
SHL <i>reg/mem16</i> , CL	D3 /4	Multiply a 16-bit register or memory operand by two the number of times specified in the CL register.
SHL <i>reg/mem16</i> , <i>imm8</i>	D1 /4 <i>ib</i>	Multiply a 16-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHL <i>reg/mem32</i> , 1	D1 /4	Multiply a 32-bit register or memory operand by two.
SHL <i>reg/mem32</i> , CL	D3 /4	Multiply a 32-bit register or memory operand by two the number of times specified in the CL register.
SHL <i>reg/mem32</i> , <i>imm8</i>	C1 /4 <i>ib</i>	Multiply a 32-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHL <i>reg/mem64</i> , 1	D1 /4	Multiply a 64-bit register or memory operand by two.
SHL <i>reg/mem64</i> , CL	D3 /4	Multiply a 64-bit register or memory operand by two the number of times specified in the CL register.
SHL <i>reg/mem64</i> , <i>imm8</i>	C1 /4 <i>ib</i>	Multiply a 64-bit register or memory operand by two the number of times specified by an 8-bit immediate value.

Related Instructions

SHR

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SHLD**Shift Left Double**

Shifts the contents of a register or memory location to the left a specified number of bits, while shifting in a bit pattern. This instruction takes three arguments. The first operand specifies the register or memory location containing the value to be shifted; the second operand specifies the register holding the bit string to shift in from the right; and the third operand specifies the 8-bit value representing the number of bits to shift to the left. This instruction only uses bits 0–4 of the count byte, thereby restricting the shift to a value between 0 and 31.

The carry flag in the rFLAGS register is set to the value of the last bit shifted out of the destination register. The auxiliary flag (AF) and the overflow flag (OF) are undefined except the following cases:

- If the count is 1, the OF flag is set to 1 when the sign of destination operand is changed (as in all other one-bit shift and rotate instructions).
- If the count is 0, all flags remain unchanged.

In legacy mode, the count mask in CL is 5 bits wide (limiting shifts to 32 bits). In 64-bit mode, the count mask in the CL register is 6 bits wide. If the count specified in CL is greater than the operand size, the result in the destination register is undefined.

Mnemonic	Opcode	Description
SHLD <i>reg/mem16, reg16, imm8</i>	0F A4	Shift bits of a 16-bit destination register or memory operand to the left the number of bits specified in an 8-bit immediate value and shift bits out of a 16-bit register into the destination operand from the right.
SHLD <i>reg/mem16, reg16, CL</i>	0F A5	Shift bits of a 16-bit destination register or memory operand to the left the number of bits specified in the CL register and shift bits out of a 16-bit register into the destination operand from the right.
SHLD <i>reg/mem32, reg32, imm8</i>	0F A4	Shift bits of a 32-bit destination register or memory operand to the left the number of bits specified in an 8-bit immediate value and shift bits out of a 32-bit register into the destination operand from the right.
SHLD <i>reg/mem32, reg32, CL</i>	0F A5	Shift bits of a 32-bit destination register or memory operand to the left the number of bits specified in the CL register and shift bits out of a 32-bit register into the destination operand from the right.
SHLD <i>reg/mem64, reg64, imm8</i>	0F A4	Shift bits of a 64-bit destination register or memory operand to the left the number of bits specified in an 8-bit immediate value and shift bits out of a 64-bit register into the destination operand from the right.

SHLD *reg/mem64, reg64, CL*

OF A5

Shift bits of a 64-bit destination register or memory operand to the left the number of bits specified in the CL register and shift bits out of a 64-bit register into the destination operand from the right.

Action

```

cnt = mod(COUNT,64);
sz = Sizeof(DEST);
IF (cnt == 0) (NOP)
else {
    if (cnt >= sz) then
    {
        // bad parameters
        // DEST is undefined;
        // CF, OF, SF, ZF, AF, PF are undefined;
    }else{ // Perform the shift
        CF = BIT[DEST, cnt - 1]; (* last bit shifted out on exit *)
        for (i = (sz - 1); i >= cnt; i--) DEST[i] = DEST[i-cnt];
        for (i = cnt-1; i >= 0; i--) DEST[i] = inBits[i-cnt +sz];
    }
}

```

Related Instructions

SHRD, SAL, SAR, SHR, SHL

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SHR Shift Right

Shifts a destination register or memory operand to the right the number of bits specified by the second operand. As bits are shifted to the left, the least-significant bit is shifted into the carry flag (CF). Most-significant bits are cleared to zero. The count operand can be an immediate value or register CL. The count is masked to 5 bits, which limits the count range to 0 to 31. A special opcode encoding is provided for a count of 1. In 64-bit mode, the count mask is 6 bits wide.

The overflow flag (OF) is set equal to the most significant bit of the original operand on 1-bit shifts. The SF, ZF, and PF flags are set according to the result. If the shift count is 0, no flags are affected.

This instruction is used to perform unsigned division by powers of two.

Mnemonic	Opcode	Description
SHR <i>reg/mem8</i> , 1	D0 /5	Unsigned divide an 8-bit register or memory operand by two.
SHR <i>reg/mem8</i> , CL	D2 /5	Unsigned divide an 8-bit register or memory operand by two the number of times specified in the CL register.
SHR <i>reg/mem8</i> , <i>imm8</i>	C0 /5 <i>ib</i>	Unsigned divide an 8-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHR <i>reg/mem16</i> , 1	D1 /5	Unsigned divide a 16-bit register or memory operand by two.
SHR <i>reg/mem16</i> , CL	D3 /5	Unsigned divide a 16-bit register or memory operand by two the number of times specified in the CL register.
SHR <i>reg/mem16</i> , <i>imm8</i>	C1 /5 <i>ib</i>	Unsigned divide a 16-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHR <i>reg/mem32</i> , 1	D1 /5	Unsigned divide a 32-bit register or memory operand by two.
SHR <i>reg/mem32</i> , CL	D3 /5	Unsigned divide a 32-bit register or memory operand by two the number of times specified in the CL register.
SHR <i>reg/mem32</i> , <i>imm8</i>	C1 /5 <i>ib</i>	Unsigned divide a 32-bit register or memory operand by two the number of times specified by an 8-bit immediate value.
SHR <i>reg/mem64</i> , 1	D1 /5	Unsigned divide a 64-bit register or memory operand by two.

SHR <i>reg/mem64</i> , CL	D3 /5	Unsigned divide a 64-bit register or memory operand by two the number of times specified in the CL register.
SHR <i>reg/mem64</i> , <i>imm8</i>	C1 /5 <i>ib</i>	Unsigned divide a 64-bit register or memory operand by two the number of times specified by an 8-bit immediate value.

Related Instructions

SHL, SHR, SAL, SAR, SHLD, SHRD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SHRD**Shift Right Double**

Shifts the contents of a register or memory to the right a specified number of bits while shifting a specified number of bits in from the left. This instruction takes three arguments. The first operand specifies the location of the value to be shifted; the second operand specifies the bit string to shift in from the left—starting with the most-significant bit of the first operand; and the third operand is either an immediate value or the contents of the CL register and specifies the number of bits to shift to the right.

The carry flag in the rFLAGS register is set to the value of the last bit shifted out of the destination register. The auxiliary flag (AF) and the overflow flag (OF) are undefined except the following cases:

- If the count is 1, the OF flag is set to 1 when the sign of destination operand is changed (as in all other one-bit shift and rotate instructions).
- If the count is 0, all flags remain unchanged.

In legacy mode, the count mask in CL is 5 bits wide (limiting shifts to 32 bits). In 64-bit mode, the count mask in the CL register is 6 bits wide. If the count specified in CL is greater than the operand size, the result in the destination register is undefined.

Mnemonic	Opcode	Description
<i>SHRD reg/mem16, reg16, imm8</i>	0F AC	Shift bits of a 16-bit destination register or memory operand to the right the number of bits specified in an 8-bit immediate value and shift bits out of a 16-bit register into the destination operand from the left.
<i>SHRD reg/mem16, reg16, CL</i>	0F AD	Shift bits of a 16-bit destination register or memory operand to the right the number of bits specified in the CL register and shift bits out of a 16-bit register into the destination operand from the left.
<i>SHRD reg/mem32, reg32, imm8</i>	0F AC	Shift bits of a 32-bit destination register or memory operand to the right the number of bits specified in an 8-bit immediate value and shift bits out of a 32-bit register into the destination operand from the left.
<i>SHRD reg/mem32, reg32, CL</i>	0F AD	Shift bits of a 32-bit destination register or memory operand to the right the number of bits specified in the CL register and shift bits out of a 32-bit register into the destination operand from the left.
<i>SHRD reg/mem16, reg64, imm8</i>	0F AC	Shift bits of a 64-bit destination register or memory operand to the right the number of bits specified in an 8-bit immediate value and shift bits out of a 64-bit register into the destination operand from the left.

SHRD *reg/mem64, reg64, CL*

OF AD

Shift bits of a 64-bit destination register or memory operand to the right the number of bits specified in the CL register and shift bits out of a 64-bit register into the destination operand from the left.

Action

```

cnt = mod(COUNT,64);
sz = Sizeof(DEST);
IF (cnt == 0) (NOP)
else {
    if (cnt >= sz) then
    {
        // bad parameters
        // DEST is undefined;
        // CF, OF, SF, ZF, AF, PF are undefined;
    }else{ (* Perform the shift *)
        CF = BIT[DEST, cnt - 1]; (* last bit shifted out on exit *)
        for (i = 0; i < (sz - cnt - 1); i++) DEST[i - cnt] = DEST[i];
        for (i = (sz - cnt); i < sz - 1; i++) DEST[i] = inBits[i+cnt -sz];
    }
}

```

Related Instructions

SHLD, SAL, SAR, SHR, SHL

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

STC Set Carry Flag

Sets the carry flag (CF) to 1 in the rFLAGS register.

The OF, ZF, SF, AF, DF, and PF flags are unaffected.

Mnemonic	Opcode	Description
STC	F9	Set carry flag.

Related Instructions

CLC.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
																1
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

None.

STD Set Direction Flag

Sets DF to 1 in the RFLAGS register. Used widely with string instructions. If DF = 1 during the execution of a string instruction, string pointers are decremented. If DF = 0, string pointers are incremented.

With the exception of I/O string instructions, all string operations use rSI as the source-operand pointer and rDI as the destination-operand pointer. I/O string instructions use rDX to specify the input-port or output-port number. For repeated string operations (those preceded with a repeat-instruction prefix), the rSI and rDI registers are incremented or decremented as the string elements are moved from the source location to the destination. Repeat-string operations also use rCX to hold the string length, and decrement it as data is moved from one location to the other.

The CF, OF, ZF, SF, AF, and PF flags are unaffected.

Mnemonic	Opcode	Description
STD	FD	Set the direction flag.

Related Instructions

CLD, INS_x , LODS_x , MOVS_x , OUTS_x , SCAS_x , STOS_x .

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
									1							
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

None.

STOS Store String

STOSB

STOSW

STOSD

STOSQ

Copies a byte, word, doubleword, or quadword from AL, AX, EAX, or RAX to a location pointed to by rDI and updates eDI to point to the next string element. The pointer DI is incremented if DF is zero; otherwise, it is decremented. The increment/decrement value is 1,2,4, or 8 depending on the number of stored bytes.

The forms of the STOSx instruction that have an explicit operand are equivalent to the forms with implicit operand (in fact, they have the same opcodes). The explicit forms, however, only allow specification of the operand size. The address of the explicit operand is otherwise ignored, and instead, the implicit operand—[rDI]—is always used.

The REP prefix can be used with the STOSx instruction. For details about the REP prefix, see Section 1.2.6, “Repeat Prefixes”, on page 10. This provides a convenient way to initialize memory blocks to a constant value. The STOSx instruction can also be put inside a loop controlled by a LOOPcc instruction to prepare values stored in memory blocks.

Mnemonic	Opcode	Description
STOS <i>reg8</i>	AA	Store the contents of the AL register to DI.
STOS <i>reg16</i>	AB	Store the contents of the AX register to DI.
STOS <i>reg32</i>	AB	Store the contents of the EAX register to EDI.
STOS <i>reg64</i>	AB	Store the contents of the RAX register to RDI.
STOSB	AA	Store the contents of the AL register to DI.
STOSW	AB	Store the contents of the AX register to DI.
STOSD	AB	Store the contents of the EAX register to EDI.
STOSQ	AB	Store the contents of the RAX register to RDI.

Related Instructions

LEA, LDS, LES, LFS, LGS, LSS.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	The memory operand effective address exceeded the limit of the ES segment.
			X	The destination operand is in a nonwritable segment.
			X	The ES register contains a null segment selector.
General protection, segment overrun, #GP	X	X		The memory operand effective address exceeded the limit of the ES segment.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

SUB**Subtract**

Subtracts source from destination and puts the result in the destination.

The destination operand can be a register or a memory location; the source operand can be an immediate, register, or memory location. When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format. Sets EFLAGS according to the result of the operation.

The processor performs the following steps for subtraction:

1. Takes the twos complement of the source,
2. Adds this to the destination, and
3. Inverts the carry and changes the flags accordingly.

The SUB instruction evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate a borrow in the signed or unsigned result, respectively. the SF flag indicates the sign of the signed result.

The LOCK prefix can be used with forms of the SUB instruction that write a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
SUB AL, <i>imm8</i>	2C <i>ib</i>	Subtract an immediate 8-bit value from the AL register and store the result in AL.
SUB AX, <i>imm16</i>	2D <i>iw</i>	Subtract an immediate 16-bit value from the AX register and store the result in AX.
SUB EAX, <i>imm32</i>	2D <i>id</i>	Subtract an immediate 32-bit value from the EAX register and store the result in EAX.
SUB RAX, <i>imm32</i>	2D <i>id</i>	Subtract an immediate 32-bit value from the RAX register and store the result in RAX.
SUB <i>reg/mem8</i> , <i>imm8</i>	80 /5 <i>ib</i>	Subtract an immediate 8-bit value from an 8-bit destination register.
SUB <i>reg/mem16</i> , <i>imm16</i>	81 /5 <i>iw</i>	Subtract an immediate 16-bit value from an 16-bit destination register or memory operand.
SUB <i>reg/mem32</i> , <i>imm32</i>	81 /5 <i>id</i>	Subtract an immediate 32-bit value from an 32-bit destination register or memory operand.

Mnemonic	Opcode	Description
<i>SUB reg/mem64, imm32</i>	81 /5 <i>id</i>	Subtract an immediate 32-bit value from an 64-bit destination register or memory operand.
<i>SUB reg/mem16, imm8</i>	83 /5 <i>ib</i>	Subtract a sign-extended immediate 8-bit value from a 16-bit register or memory operand.
<i>SUB reg/mem32, imm8</i>	83 /5 <i>ib</i>	Subtract a sign-extended immediate 8-bit value from a 32-bit register or memory operand.
<i>SUB reg/mem64, imm8</i>	83 /5 <i>ib</i>	Subtract a sign-extended immediate 8-bit value from a 64-bit register or memory operand.
<i>SUB reg/mem8, reg8</i>	28 / <i>r</i>	Subtract the contents of an 8-bit register from an 8-bit destination register.
<i>SUB reg/mem16, reg16</i>	29 / <i>r</i>	Subtract the contents of a 16-bit register from a 16-bit destination register.
<i>SUB reg/mem32, reg32</i>	29 / <i>r</i>	Subtract the contents of an 32-bit register from an 32-bit destination register.
<i>SUB reg/mem64, reg64</i>	29 / <i>r</i>	Subtract the contents of an 64-bit register from an 64-bit destination register.
<i>SUB reg8, reg/mem8</i>	2A / <i>r</i>	Subtract the contents of an 8-bit register or memory operand from an 8-bit destination register.
<i>SUB reg16, reg/mem16</i>	2B / <i>r</i>	Subtract the contents of a 16-bit register or memory operand from a 16-bit destination register.
<i>SUB reg32, reg/mem32</i>	2B / <i>r</i>	Subtract the contents of a 32-bit register or memory operand from a 32-bit destination register.
<i>SUB reg64, reg/mem64</i>	2B / <i>r</i>	Subtract the contents of a 64-bit register or memory operand from a 64-bit destination register.

Related Instructions

ADC, ADD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

TEST**Test Bits**

Performs a bit-wise logical AND on the two operands, setting flags but leaving the contents of both source and destination unchanged. While the AND instruction changes the contents of the destination and the flag bits, the TEST instruction changes only the flag bits as shown in the **ACTION** section.

Mnemonic	Opcode	Description
TEST AL, <i>imm8</i>	A8 <i>ib</i>	AND an immediate 8-bit value with the contents of the AL register and set EFLAGS to reflect the result.
TEST AX, <i>imm16</i>	A9 <i>iw</i>	AND an immediate 16-bit value with the contents of the AX register and set EFLAGS to reflect the result.
TEST EAX, <i>imm32</i>	A9 <i>id</i>	AND an immediate 32-bit value with the contents of the EAX register and set EFLAGS to reflect the result.
TEST RAX, <i>imm32</i>	A9 <i>id</i>	AND an immediate 32-bit value with the contents of the RAX register and set EFLAGS to reflect the result.
TEST <i>reg/mem8</i> , <i>imm8</i>	F6 /0 <i>ib</i>	AND an immediate 8-bit value with the contents of an 8-bit register or memory operand and set EFLAGS to reflect the result.
TEST <i>reg/mem16</i> , <i>imm16</i>	F7 /0 <i>iw</i>	AND an immediate 32-bit value with the contents of a 16-bit register or memory operand and set EFLAGS to reflect the result.
TEST <i>reg/mem32</i> , <i>imm32</i>	F7 /0 <i>id</i>	AND an immediate 32-bit value with the contents of a 32-bit register or memory operand and set EFLAGS to reflect the result.
TEST <i>reg/mem64</i> , <i>imm32</i>	F7 /0 <i>id</i>	AND an immediate 32-bit value with the contents of a 64-bit register or memory operand and set the SF, ZF, and PF flags to reflect the result.
TEST <i>reg/mem8</i> , <i>reg8</i>	84 / <i>r</i>	AND the contents of an 8-bit register with the contents of an 8-bit register or memory operand and set the SF, ZF, and PF flags to reflect the result.
TEST <i>reg/mem16</i> , <i>reg16</i>	85 / <i>r</i>	AND the contents of a 16-bit register with the contents of a 16-bit register or memory operand and set EFLAGS to reflect the result.
TEST <i>reg/mem32</i> , <i>reg32</i>	85 / <i>r</i>	AND the contents of a 32-bit register with the contents of a 32-bit register or memory operand and set EFLAGS to reflect the result.
TEST <i>reg/mem64</i> , <i>reg64</i>	85 / <i>r</i>	AND the contents of a 64-bit register with the contents of a 64-bit register or memory operand and set EFLAGS to reflect the result.

Action

```

for (i = 0; i < sizeof (src1); i++) tmp[i] = src1[i] && src2[i];
SF = msb(tmp);
if (tmp == 0) ZF = 0;
else ZF = 1;
PF = BitwiseXNOR(tmp[0:7]);
CF = 0;
OF = 0;

```

Related Instructions

AND, CMP, SUB

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0	1/0	U	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

XADD Exchange and Add

Exchanges the source operand with the destination operand, then loads the sum of the two values into the destination operand. The destination operand can be a register or a memory location; the source operand is a register.

The LOCK prefix can be used with forms of the XADD instruction that use a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
<i>XADD reg/mem8, reg8</i>	0F C0 /r	Exchange the contents of an 8-bit register and an 8-bit destination register or memory operand and load their sum into the destination.
<i>XADD reg/mem16, reg16</i>	0F C1 /r	Exchange the contents of a 16-bit register and a 16-bit destination register or memory operand and load their sum into the destination.
<i>XADD reg/mem32, reg32</i>	0F C1 /r	Exchange the contents of a 32-bit register and a 32-bit destination register or memory operand and load their sum into the destination.
<i>XADD reg/mem64, reg64</i>	0F C1 /r	Exchange the contents of a 64-bit register and a 64-bit destination register or memory operand and load their sum into the destination.

Related Instructions

XCHG, ADD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								1/0				1/0		1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

XCHG Exchange

Exchanges the contents of the destination and source operands. The operands can be two general-purpose registers or a register and a memory location. If a memory operand is referenced, the locking protocol of the processor is automatically implemented regardless of the presence or absence of the LOCK prefix or of the value of IOPL.

For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

The XCHG instruction must be used instead of the BSWAP instruction for 16-bit operands.

Mnemonic	Opcode	Description
XCHG AX, <i>reg16</i>	90 <i>+rw</i>	Exchange the contents of the AX register with the contents of a 16-bit register.
XCHG <i>reg16</i> , AX	90 <i>+rw</i>	Exchange the contents of a 16-bit register with the contents of the AX register.
XCHG EAX, <i>reg32</i>	90 <i>+rd</i>	Exchange the contents of the EAX register with the contents of a 32-bit register.
XCHG <i>reg32</i> , EAX	90 <i>+rd</i>	Exchange the contents of a 32-bit register with the contents of the EAX register.
XCHG RAX, <i>reg64</i>	90 <i>+rq</i>	Exchange the contents of the RAX register with the contents of a 64-bit register.
XCHG <i>reg64</i> , RAX	90 <i>+rq</i>	Exchange the contents of a 64-bit register with the contents of the RAX register.
XCHG <i>reg/mem8</i> , <i>reg8</i>	86 <i>/r</i>	Exchange the contents of an 8-bit register with the contents of an 8-bit register or memory operand.
XCHG <i>reg8</i> , <i>reg/mem8</i>	86 <i>/r</i>	Exchange the contents of an 8-bit register or memory operand with the contents of an 8-bit register.
XCHG <i>reg/mem16</i> , <i>reg16</i>	87 <i>/r</i>	Exchange the contents of a 16-bit register with the contents of a 16-bit register or memory operand.
XCHG <i>reg16</i> , <i>reg/mem16</i>	87 <i>/r</i>	Exchange the contents of a 16-bit register or memory operand with the contents of a 16-bit register.
XCHG <i>reg/mem32</i> , <i>reg32</i>	87 <i>/r</i>	Exchange the contents of a 32-bit register with the contents of a 32-bit register or memory operand.
XCHG <i>reg32</i> , <i>reg/mem32</i>	87 <i>/r</i>	Exchange the contents of a 32-bit register or memory operand with the contents of a 32-bit register.

XCHG *reg/mem64, reg64* 87 /r

Exchange the contents of a 64-bit register with the contents of a 64-bit register or memory operand.

XCHG *reg64, reg/mem64* 87 /r

Exchange the contents of a 64-bit register or memory operand with the contents of a 64-bit register.

The x86 architecture commonly uses the XCHG EAX, EAX instruction (opcode 90h) as a one-byte NOP. In 64-bit mode, the processor treats opcode 90h as a true NOP only if it would exchange rAX with itself. Opcode 90h can still be used to exchange RAX and another 64-bit register (e.g., R8) if the appropriate REX prefix is used. Without this special handling, the instruction would zero-extend the upper-32 bits of RAX, and thus it would not be a true no-operation.

This special handling does not apply to the two-byte ModRM form of the XCHG instruction.

Related Instructions

BSWAP

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

XLAT

XLATB

Translate Table Index

Locates a byte entry in a table in memory, using the contents of the AL register as a table index, then copies the contents of the table entry back into the AL register.

The index in the AL register is treated as an unsigned integer. The XLATB instruction gets the base address of the table in memory from either the DS:rBX register pair. (The DS segment may be overridden with a segment override prefix.) Often used to translate data from one format to another, such as ASCII to EBCDIC.

This instruction writes AL without changing RAX[63:08]. This instruction ignores operand size.

The assembler uses the single operand to document the segment and addressing size attribute. The effective address is always specified by the rBX registers.

Mnemonic	Opcode	Description
XLAT <i>mem8</i>	D7	Set the contents AL to the contents of DS:[rBX + unsigned AL].
XLATB	D7	Set the contents AL to the contents of DS:[rBX + unsigned AL].

Related Instructions

XLATB

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

XOR Logical Exclusive OR

Performs a bitwise exclusive OR (XOR) operation on the destination and source operands and stores the result in the destination operand location. The source operand can be an immediate value, a register, or a memory location; the destination operand can be a register or a memory location. `XOR AX, AX` can be used to clear AX.

The LOCK prefix can be used with forms of the XOR instruction that write a memory operand. For details about the LOCK prefix, see Section 1.2.5, “Lock Prefix”, on page 10.

Mnemonic	Opcode	Description
<code>XOR AL, imm8</code>	<code>34 ib</code>	XOR the contents of AL and an immediate 8-bit operand and store the results in AL.
<code>XOR AX, imm16</code>	<code>35 iw</code>	XOR the contents of AX and an immediate 16-bit operand and store the results in AX.
<code>XOR EAX, imm32</code>	<code>35 id</code>	XOR the contents of EAX and an immediate 32-bit operand and store the results in EAX.
<code>XOR RAX, imm32</code>	<code>35 id</code>	XOR the contents of RAX and an immediate 32-bit operand and store the results in RAX.
<code>XOR reg/mem8, imm8</code>	<code>80 /6 ib</code>	XOR the contents of an 8-bit destination register or memory operand with an 8-bit immediate value and store the result in the destination.
<code>XOR reg/mem16, imm16</code>	<code>81 /6 iw</code>	XOR the contents of a 16-bit destination register or memory operand with a 16-bit immediate value and store the results in the destination.
<code>XOR reg/mem32, imm32</code>	<code>81 /6 id</code>	XOR the contents of a 32-bit destination register or memory operand with a 32-bit immediate value and store the results in the destination.
<code>XOR reg/mem64, imm32</code>	<code>81 /6 id</code>	XOR the contents of a 64-bit destination register or memory operand with a 32-bit immediate value and store the results in the destination.
<code>XOR reg/mem16, imm8</code>	<code>83 /6 ib</code>	XOR the contents of a 16-bit destination register or memory operand and an 8-bit immediate value and store the results in the destination.
<code>XOR reg/mem32, imm8</code>	<code>83 /6 ib</code>	XOR the contents of a 32-bit destination register or memory operand and an 8-bit immediate value and store the results in the destination.
<code>XOR reg/mem64, imm8</code>	<code>83 /6 ib</code>	XOR the contents of a 64-bit destination register or memory operand and an 8-bit immediate value and store the results in the destination.
<code>XOR reg/mem8, reg8</code>	<code>30 /r</code>	XOR the contents of an 8-bit destination register with the contents of an 8-bit register or memory operand and store the contents in the destination.
<code>XOR reg/mem16, reg16</code>	<code>31 /r</code>	XOR the contents of a 16-bit destination register with the contents of a 16-bit register or memory operand and store the contents in the destination.

Mnemonic	Opcode	Description
<i>XOR reg/mem32, reg32</i>	31 /r	XOR the contents of a 32-bit destination register with the contents of a 32-bit register or memory operand and store the contents in the destination.
<i>XOR reg/mem64, reg64</i>	31 /r	XOR the contents of a 64-bit destination register with the contents of a 64-bit register or memory operand and store the contents in the destination.
<i>XOR reg8, reg/mem8</i>	32 /r	XOR the contents of an 8-bit destination register with the contents of an 8-bit register or memory operand and store the results in the destination.
<i>XOR reg16, reg/mem16</i>	33 /r	XOR the contents of a 16-bit destination register with the contents of a 16-bit register or memory operand and store the results in the destination.
<i>XOR reg32, reg/mem32</i>	33 /r	XOR the contents of a 32-bit destination register with the contents of a 32-bit register or memory operand and store the results in the destination.
<i>XOR reg64, reg/mem64</i>	33 /r	XOR the contents of a 64-bit destination register with the contents of a 64-bit register or memory operand and store the results in the destination.

The instructions performs the following operation for each bit:

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Related Instructions

OR, AND, NOT

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				1/0	1/0	U	1/0	0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

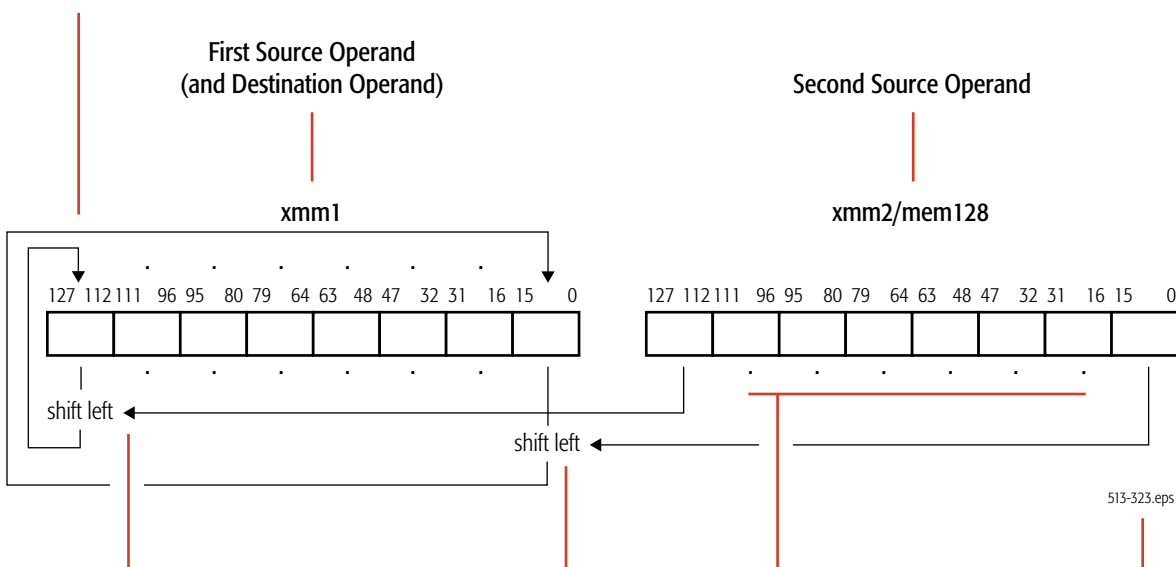
Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

2.7 128-Bit Media Instruction Reference

This section describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by the 128-bit media instructions. These instructions load, store, or operate on data located in the 128-bit XMM registers. Most of the instructions operate in parallel on sets of packed elements called *vectors*, although a few operate on scalars. The instructions define both integer and floating-point operations. They include the legacy SSE and SSE-2 instructions.

Each instruction that performs a vector (packed) operation is illustrated with a diagram. Figure 2-15 shows the conventions used in these diagrams. The particular diagram shows the PSLW (packed shift left logical words) instruction.

Arrowheads going to a source operand indicate the writing of the result. In this case, the result is written to the first source operand, which is also the destination operand.



Arrowheads coming from a source operand indicate that the source operand provides a *control function*. In this case, the second source operand specifies the *number* of bits to shift, and the first source operand specifies the *data* to be shifted.

Operation. In this case, a bitwise shift-left.

Ellipses indicate that the operation is repeated for each element of the source vectors. In this case, there are 8 elements in each source vector, so the operation is performed 8 times, in parallel.

File name of this figure (for documentation control)

Figure 2-15. Diagram Conventions for 128-Bit Media Instructions

Gray areas in diagrams indicate unmodified operand bits.

The 128-bit media instructions are useful in high-performance applications that operate on blocks of data. Because each instruction can independently and simultaneously perform a single operation on multiple elements of a vector, the instructions are classified as *single-instruction, multiple-data* (SIMD) instructions. A few 128-bit media instructions convert operands in XMM registers to operands in GPR, MMX™, or x87 registers (or vice versa), or save or restore XMM state.

Hardware support for a specific 128-bit media instruction depends on the presence of at least one of the following CPUID functions:

- FXSAVE and FXRSTOR, indicated by bit 24 of CPUID standard function 1 and extended function 8000_0001h.
- SSE, indicated by bit 25 of CPUID extended function 8000_0001h.
- SSE-2, indicated by bit 26 of CPUID extended function 8000_0001h.

The 128-bit media instructions can be used in legacy mode or long mode. Their use in long mode is available if the following CPUID function is set:

- Long Mode, indicated by bit 29 of CPUID extended function 8000_0001h.

Compilation of 128-bit media programs for execution in 64-bit mode offers four primary advantages: access to the eight extended XMM registers (for a register set consisting of XMM0–XMM15), access to the eight extended, 64-bit general-purpose registers (for a register set consisting of GPR0–GPR15), access to the 64-bit virtual address space, and access to the RIP-relative addressing mode.

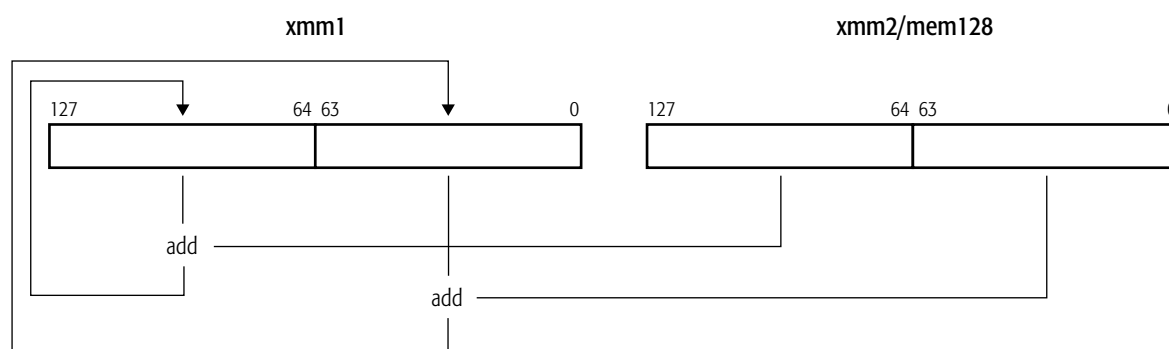
For further information, see:

- Volume 1: Section 4, “128-Bit Media and Scientific Programming”, on page 131.
- Volume 3: Section 2.3, “Summary of Registers and Data Types”, on page 28.
- Volume 3: Section 2.5, “Notation”, on page 41.
- Volume 3: Section 1.2, “Instruction Prefixes”, on page 3.

ADDPD**Add Packed Double-Precision Floating-Point**

The ADDPD instruction adds each packed double-precision floating-point value in the first source operand to the corresponding packed double-precision floating-point value in the second source operand and writes the result of each addition in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ADDPD <i>xmm1, xmm2/mem128</i>	66 0F 58 /r	Adds two packed double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



addpd.eps

Related Instructions

ADDPS, ADDSD, ADDSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

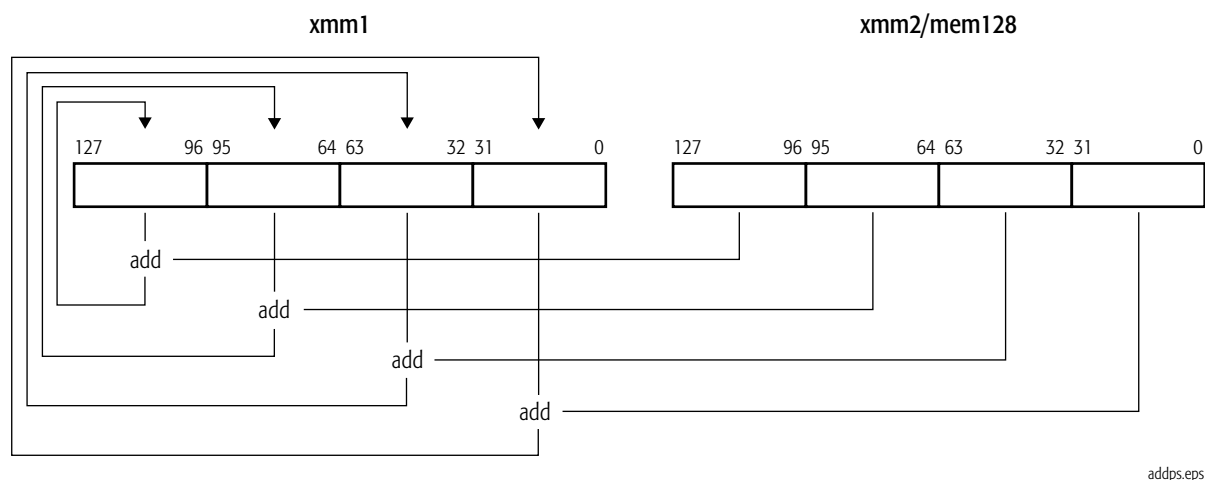
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

ADDPS**Add Packed Single-Precision Floating-Point**

The ADDPS instruction adds each packed single-precision floating-point value in the first source operand to the corresponding packed single-precision floating-point value in the second source operand and writes the result of each addition in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ADDPS <i>xmm1, xmm2/mem128</i>	OF 5B /r	Adds four packed single-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ADDPD, ADDSD, ADDSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

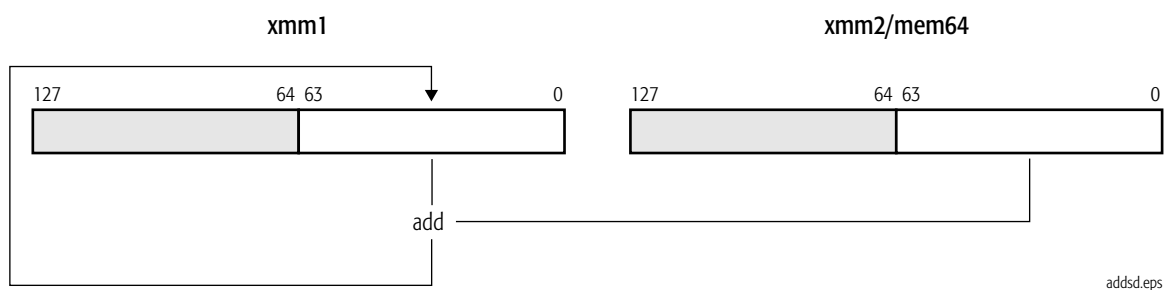
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

ADDSD**Add Scalar Double-Precision Floating-Point**

The ADDSD instruction adds the double-precision floating-point value in the low-order quadword of the first source operand to the double-precision floating-point value in the low-order quadword of the second source operand and writes the result in the low-order quadword of the destination (first source). The high-order quadword of the destination is not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 64-bit memory location.

Mnemonic	Opcode	Description
ADDSD <i>xmm1, xmm2/mem64</i>	F2 0F 58 /r	Adds low-order double-precision floating-point values in an XMM register and another XMM register or 64-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ADDPD, ADDPS, ADDSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

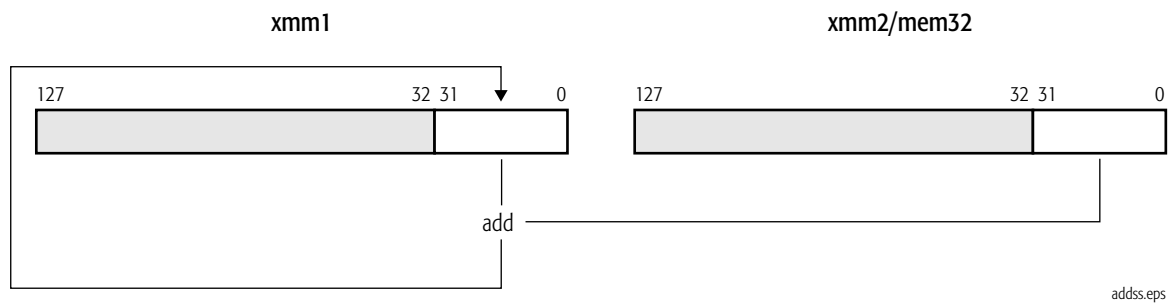
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

ADDSS**Add Scalar Single-Precision Floating-Point**

The ADDSS instruction adds the single-precision floating-point value in the low-order doubleword of the first source operand to the single-precision floating-point value in the low-order doubleword of the second source operand and writes the result in the low-order doubleword of the destination (first source). The three high-order doublewords of the destination are not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 32-bit memory location.

Mnemonic	Opcode	Description
ADDSS <i>xmm1, xmm2/mem32</i>	F3 0F 58 /r	Adds low-order single-precision floating-point values in an XMM register and another XMM register or 32-bit memory location and writes the result in the destination XMM register.



addss.eps

Related Instructions

ADDPD, ADDPS, ADDSD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

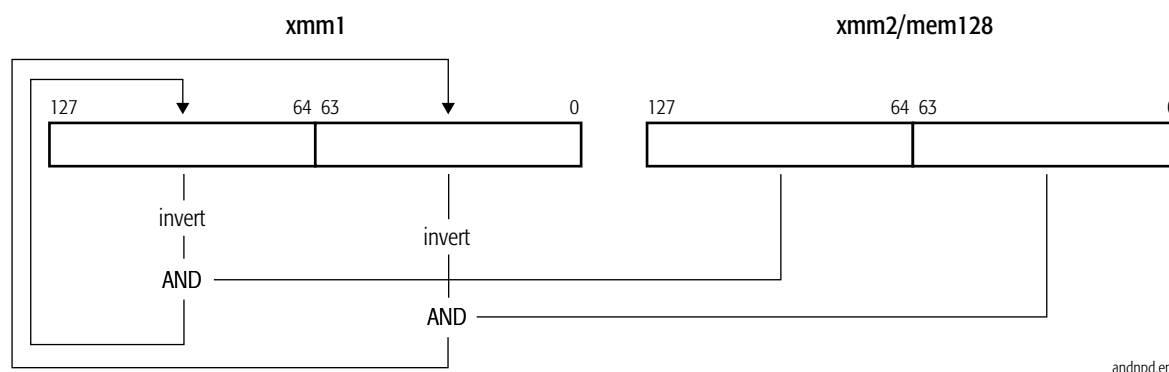
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

ANDNPD**Logical Bitwise AND NOT
Packed Double-Precision Floating-Point**

The ANDNPD instruction performs a bitwise logical AND of the two packed double-precision floating-point values in the second source operand and the one's-complement of the corresponding two packed double-precision floating-point values in the first source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ANDNPD <i>xmm1, xmm2/mem128</i>	66 0F 55 /r	Performs bitwise logical AND NOT of two packed double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



andnpd.eps

Related Instructions

ANDNPS, ANDPD, ANDPS, ORPD, ORPS, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

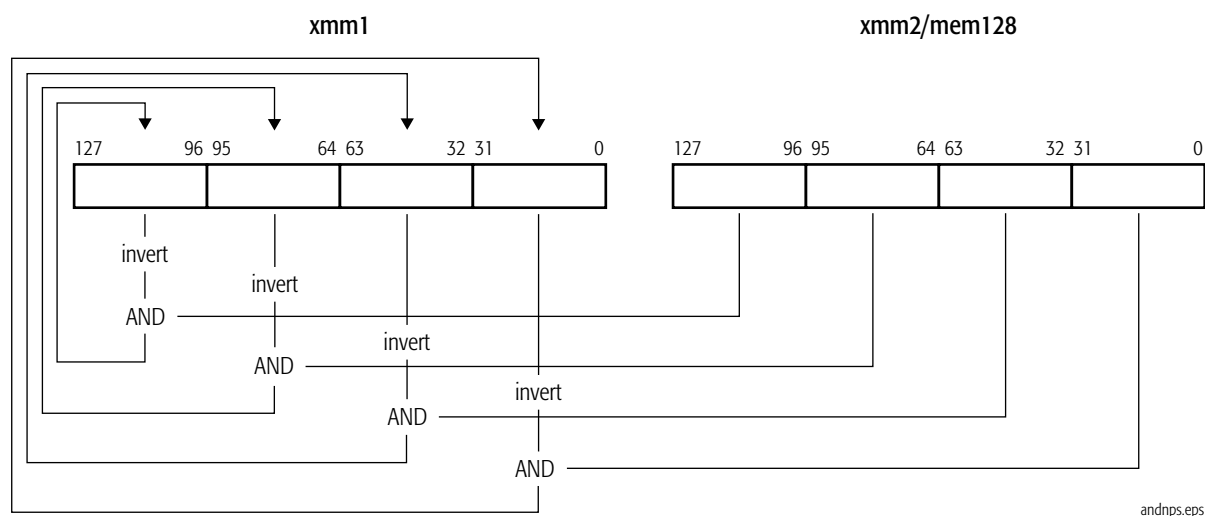
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	<p>The emulate bit (EM) of CR0 is set to 1.</p> <p>The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception.</p> <p>The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0.</p> <p>The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.</p>
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		One of the data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

ANDNPS**Logical Bitwise AND NOT
Packed Single-Precision Floating-Point**

The ANDNPS instruction performs a bitwise logical AND of the four packed single-precision floating-point values in the second source operand and the one's-complement of the corresponding four packed single-precision floating-point values in the first source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ANDNPS <i>xmm1, xmm2/mem128</i>	OF 55/ <i>r</i>	Performs bitwise logical AND NOT of four packed single-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ANDNPD, ANDPD, ANDPS, ORPD, ORPS, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

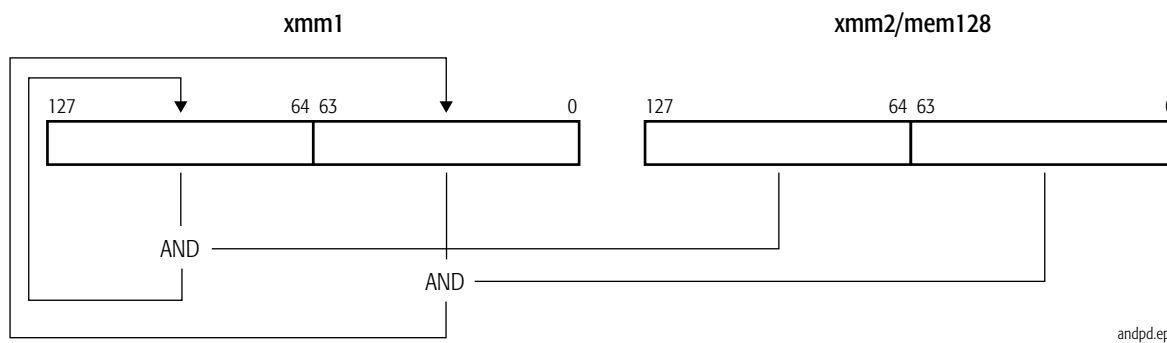
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

ANDPD**Logical Bitwise AND
Packed Double-Precision Floating-Point**

The ANDPD instruction performs a bitwise logical AND of the two packed double-precision floating-point values in the first source operand and the corresponding two packed double-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ANDPD <i>xmm1, xmm2/mem128</i>	66 0F 54 /r	Performs bitwise logical AND of two packed double-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ANDNPD, ANDNPS, ANDPS, ORPD, ORPS, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

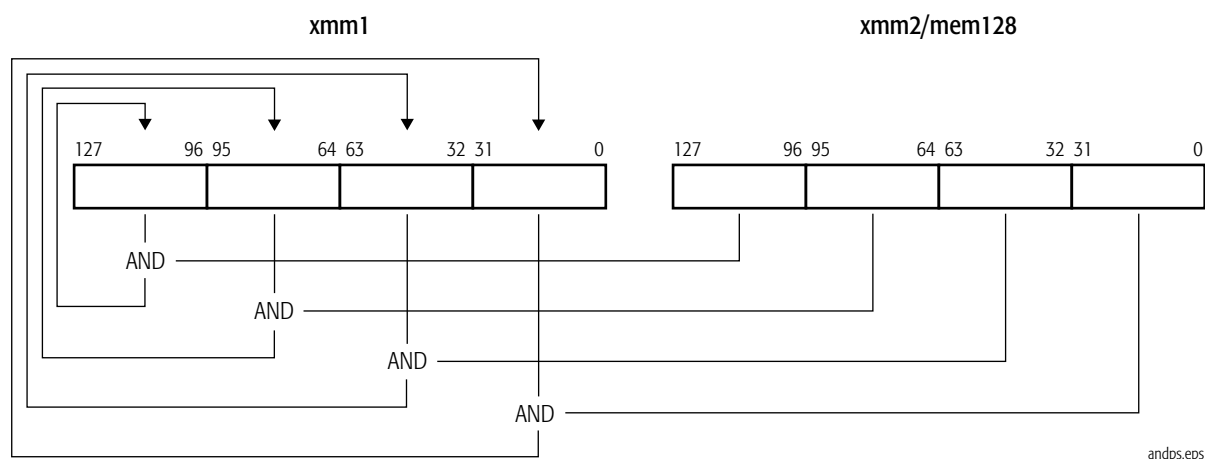
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

ANDPS**Logical Bitwise AND
Packed Single-Precision Floating-Point**

The ANDPS instruction performs a bitwise logical AND of the four packed single-precision floating-point values in the first source operand and the corresponding four packed single-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ANDPS <i>xmm1, xmm2/mem128</i>	0F 54 /r	Performs bitwise logical AND of four packed single-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



andps.eps

Related Instructions

ANDNPD, ANDNPS, ANDPD, ORPD, ORPS, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

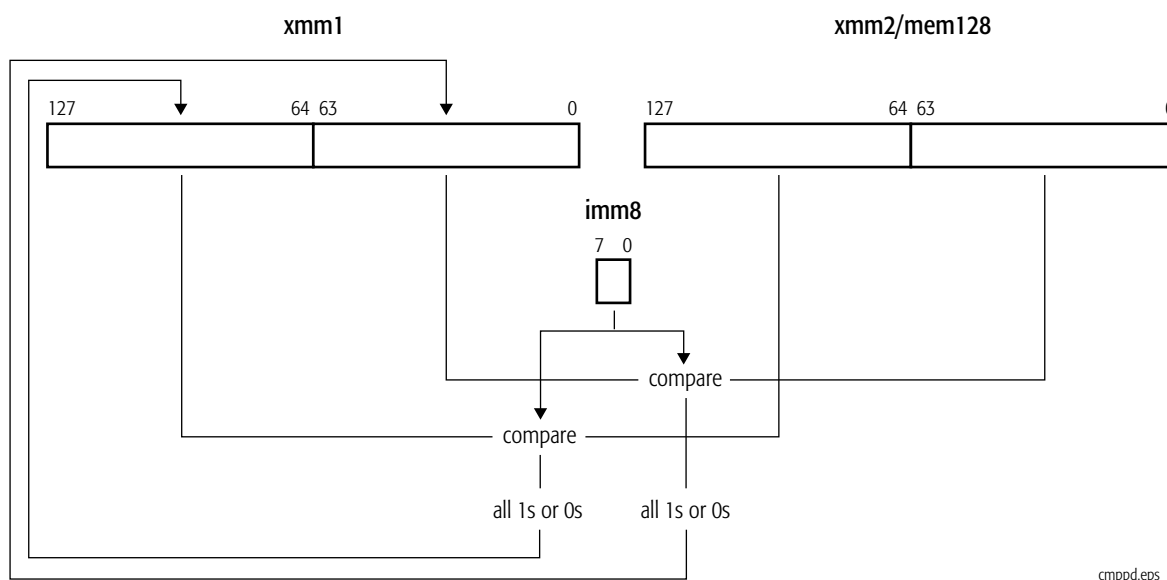
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

CMPPD**Compare Packed Double-Precision Floating-Point**

The CMPPD instruction compares each of the two packed double-precision floating-point values in the first source operand with the corresponding packed double-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding 64 bits of the destination (first source). The type of comparison is specified by the three low-order bits of the immediate-byte operand, as shown in Table 2-5. The result of each compare is a 64-bit value of all 1s (TRUE) or all 0s (FALSE). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
CMPPD <i>xmm1, xmm2/mem128, imm8</i>	66 0F C2 /r ib	Compares two pairs of packed double-precision floating-point values in an XMM register and an XMM register or 128-bit memory location.



cmppd.eps

Some compare operations that are not directly supported by the immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and then executing the appropriate compare instruction using the swapped values. These additional compare operations are shown, together with the directly supported compare operations, in Table 2-5. When swapping operands, the first source XMM register is overwritten by the result.

Table 2-5. Immediate Operand Values for Compare Operations

Immediate-Byte Value (bits 2:0)	Compare Operation	Result If NaN Operand	QNaN Operand Causes Invalid Operation Exception
000	Equal	FALSE	No
001	Less than	FALSE	Yes
	Greater than or equal to (uses swapped operands)	FALSE	Yes
010	Less than or equal	FALSE	Yes
	Greater than (uses swapped operands)	FALSE	Yes
011	Unordered	TRUE	No
100	Not equal	TRUE	No
101	Not less than	TRUE	Yes
	Not greater than (uses swapped operands)	TRUE	Yes
110	Not less than or equal	TRUE	Yes
	Not greater than or equal (uses swapped operands)	TRUE	Yes
111	Ordered	FALSE	No

Related Instructions

CMPPS, CMPSD, CMPSS, COMISD, COMISS, UCOMISD, UCOMISS

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

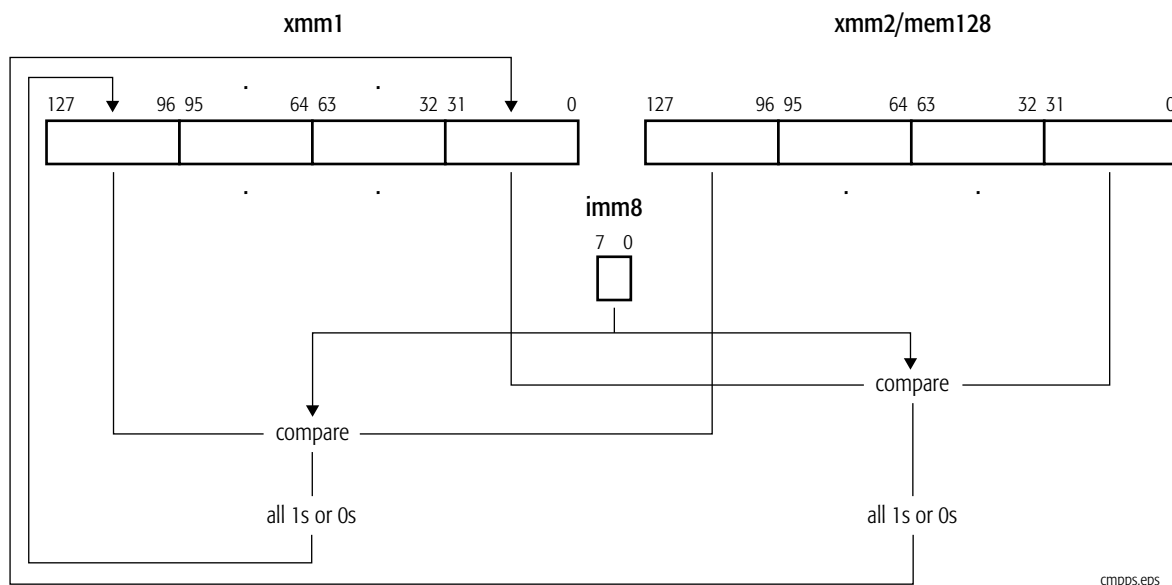
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value. For some compares, the source operand is a QNaN value (refer to Table 2-5 on page 312).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CMPPS**Compare Packed Single-Precision Floating-Point**

The CMPPS instruction compares each of the four packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding 32 bits of the destination (first source). The type of comparison is specified by the three low-order bits of the immediate-byte operand, as shown in Table 2-5 on page 312. The result of each compare is a 32-bit value of all 1s (TRUE) or all 0s (FALSE). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
CMPPS <i>xmm1, xmm2/mem128, imm8</i>	0F C2 /r ib	Compares four pairs of packed single-precision floating-point values in an XMM register and an XMM register or 64-bit memory location.



cmpps.eps

Some compare operations that are not directly supported by the immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and then executing the appropriate compare instruction using the swapped values. These additional compare operations are shown in Table 2-5 on

page 312. When swapping operands, the first source XMM register is overwritten by the result.

Related Instructions

CMPPD, CMPSD, CMPSS, COMISD, COMISS, UCOMISD, UCOMISS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

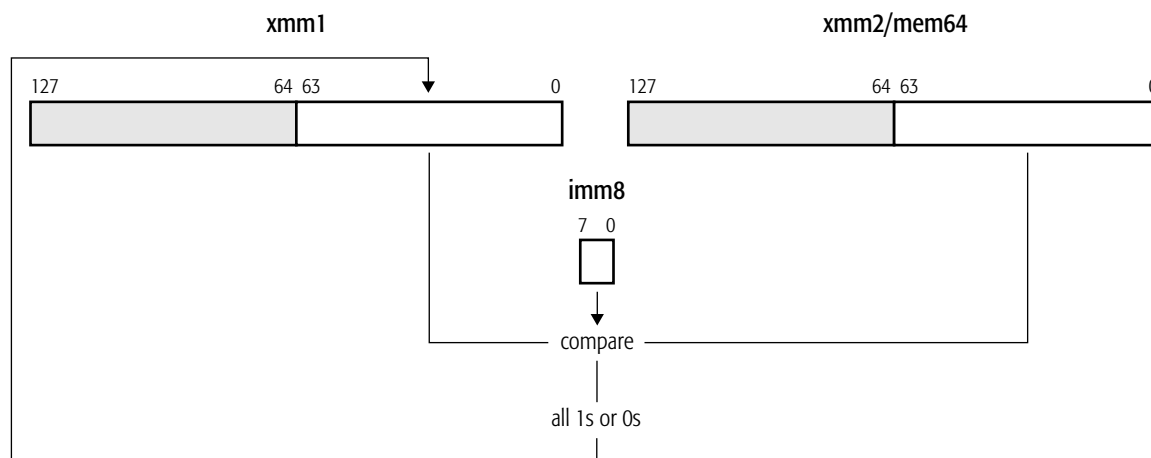
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value. For some compares, the source operand is a QNaN value (refer to Table 2-5 on page 312).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CMPSD**Compare Scalar Double-Precision Floating-Point**

The CMPSD instruction compares the double-precision floating-point value in the low-order 64 bits of the first source operand with the double-precision floating-point value in the low-order 64 bits of the second source operand and writes the result in the low-order 64 bits of the destination (first source). The type of comparison is specified by the three low-order bits of the immediate-byte operand, as shown in Table 2-5 on page 312. The result of the compare is a 64-bit value of all 1s (TRUE) or all 0s (FALSE). The first source/destination operand is an XMM register. The second source operand is another XMM register or 64-bit memory location. The high-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
CMPSD <i>xmm1, xmm2/mem64, imm8</i>	F2 0F C2 /r ib	Compares double-precision floating-point values in an XMM register and an XMM register or 64-bit memory location.



cmpsd.eps

Some compare operations that are not directly supported by the immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and then executing the appropriate compare instruction using the swapped values. These additional compare operations are shown in Table 2-5 on page 312. When swapping operands, the first source XMM register is overwritten by the result.

This CMPSD instruction should not be confused with the same-mnemonic CMPSD (compare strings by doubleword) instruction in the general-purpose instruction set. Assemblers can distinguish the instructions by the number and type of operands.

Related Instructions

CMPPD, CMPPS, CMPSS, COMISD, COMISS, UCOMISD, UCOMISS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE	
															1/0	1/0	
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

Exceptions

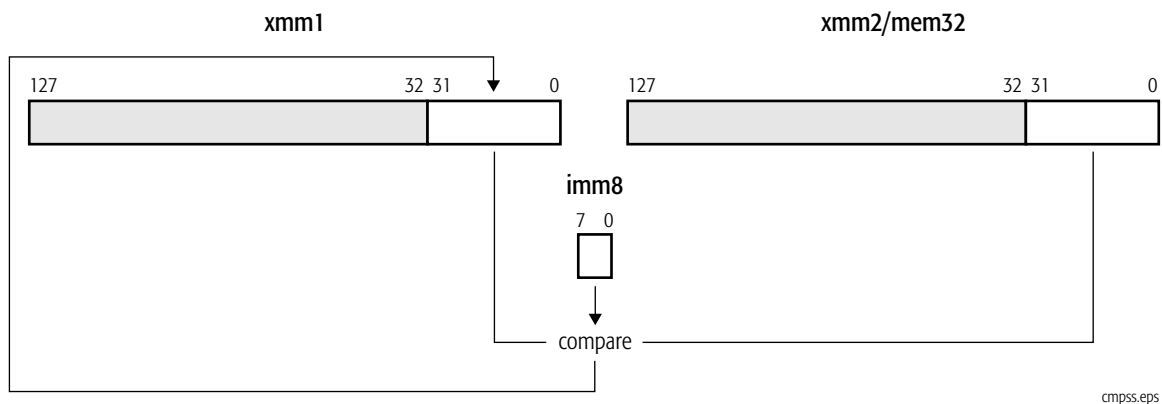
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value. For some compares, the source operand is a QNaN value (refer to Table 2-5 on page 312).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CMPSS**Compare Scalar Single-Precision Floating-Point**

The CMPSS instruction compares the single-precision floating-point value in the low-order 32 bits of the first source operand with the single-precision floating-point value in the low-order 32 bits of the second source operand and writes the result in the low-order 32 bits of the destination (first source). The type of comparison is specified by the three low-order bits of the immediate-byte operand, as shown in Table 2-5 on page 312. The result of the compare is a 32-bit value of all 1s (TRUE) or all 0s (FALSE). The first source/destination operand is an XMM register. The second source operand is another XMM register or 32-bit memory location. The three high-order doublewords of the destination XMM register are not modified.

Mnemonic	Opcode	Description
CMPSS <i>xmm1, xmm2/mem32, imm8</i>	F3 0F C2 /r ib	Compares single-precision floating-point values in an XMM register and an XMM register or 32-bit memory location.



cmpss.eps

Some compare operations that are not directly supported by the immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and then executing the appropriate compare instruction using the swapped values. These additional compare operations are shown in Table 2-5 on page 312. When swapping operands, the first source XMM register is overwritten by the result.

Related Instructions

CMPPD, CMPPS, CMPSD, COMISD, COMISS, UCOMISD, UCOMISS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

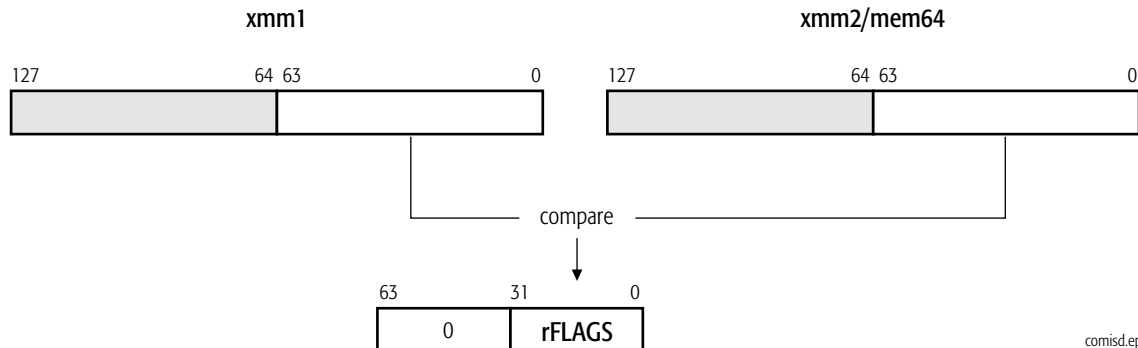
Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value. For some compares, the source operand is a QNaN value (refer to Table 2-5 on page 312).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

COMISD**Compare Ordered Scalar Double-Precision Floating-Point**

The COMISD instruction performs an ordered compare of the double-precision floating-point value in the low-order 64 bits of an XMM register with the double-precision floating-point value in the low-order 64 bits of another XMM register or a 64-bit memory location and sets the ZF, PF, and CF bits in the rFLAGS register to reflect the result of the compare. The result is unordered if one or both of the operand values is a NaN. The OF, AF, and SF bits in rFLAGS are set to zero.

If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

Mnemonic	Opcode	Description
COMISD <i>xmm1, xmm2/mem64</i>	66 0F 2F /r	Compares double-precision floating-point values in an XMM register and an XMM register or 64-bit memory location. Sets rFLAGS.



comisd.eps

Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

Related Instructions

CMPPD, CMPPS, CMPSD, CMPSS, COMISS, UCOMISD, UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	1/0	0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank.
 If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

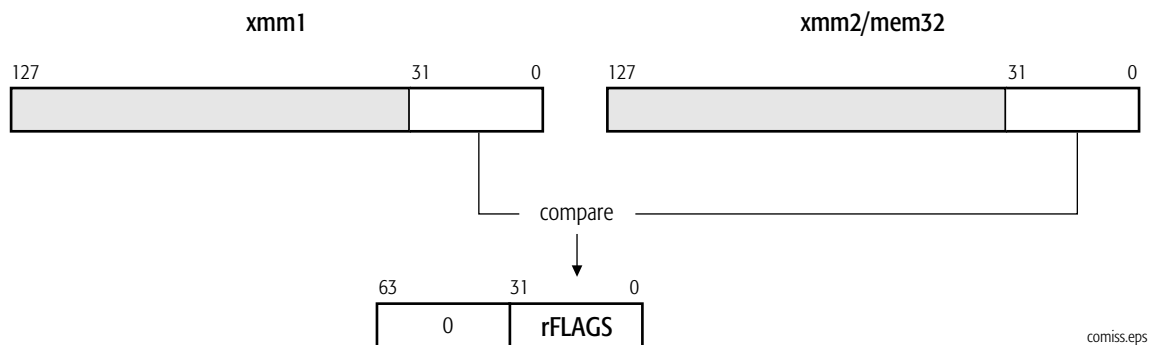
Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN or QNaN value or an unsupported format.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

COMISS**Compare Ordered Scalar Single-Precision Floating-Point**

The COMISS instruction performs an ordered compare of the single-precision floating-point value in the low-order 32 bits of an XMM register with the single-precision floating-point value in the low-order 32 bits of another XMM register or a 32-bit memory location and sets the ZF, PF, and CF bits in the rFLAGS register to reflect the result of the compare. The result is unordered if one or both of the operand values is a NaN. The OF, AF, and SF bits in rFLAGS are set to zero.

If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

Mnemonic	Opcode	Description
COMISS <i>xmm1, xmm2/mem32</i>	0F 2F /r	Compares single-precision floating-point values in an XMM register and an XMM register or 32-bit memory location. Sets rFLAGS.



comiss.eps

Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

Related Instructions

CCMPPD, CMPPS, CMPSD, CMPSS, COMISD, UCOMISD, UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	1/0	0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank.
 If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

Exceptions

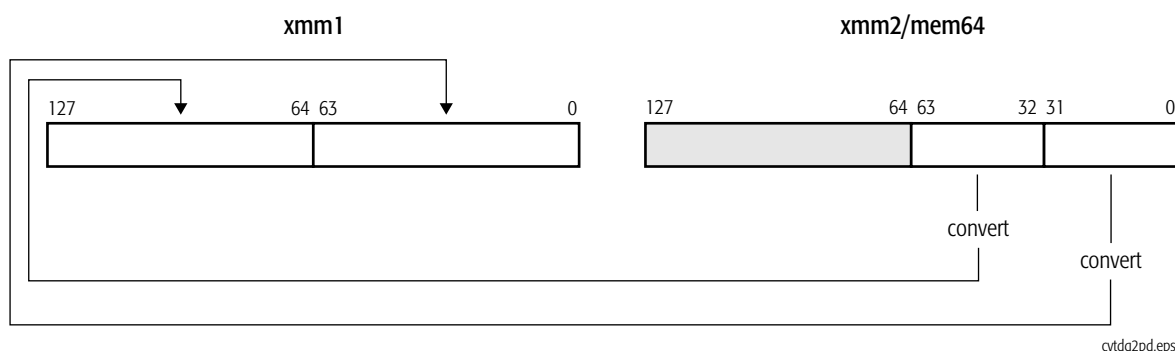
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN or QNaN value or an unsupported format.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CVTDQ2PD**Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point**

The CVTDQ2PD instruction converts two packed 32-bit signed integer values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed double-precision floating-point values and writes the converted values in another XMM register.

Mnemonic	Opcode	Description
CVTDQ2PD <i>xmm1, xmm2/mem64</i>	F3 0F E6	Converts packed doubleword signed integers in an XMM register or 64-bit memory location to double-precision floating-point values in the destination XMM register.

**Related Instructions**

CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

None.

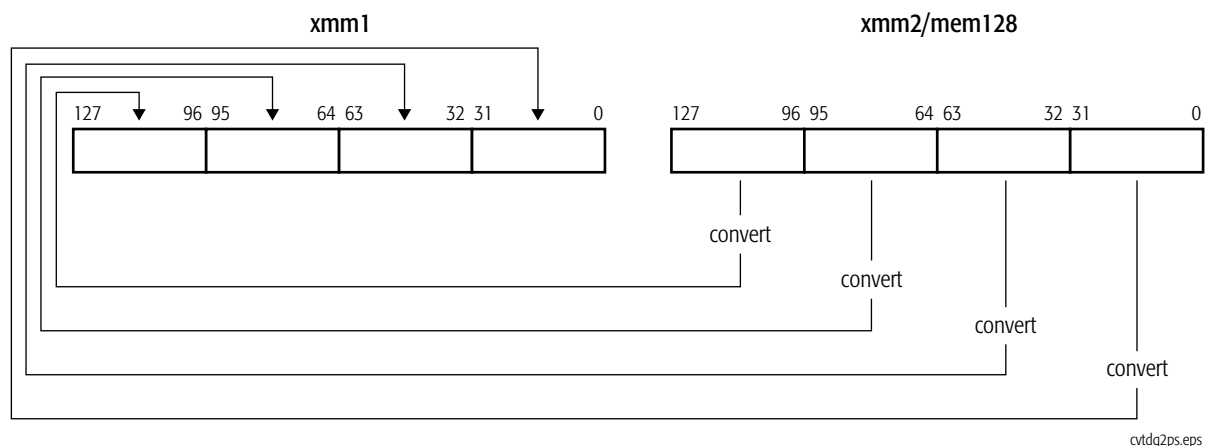
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

CVTDQ2PS**Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point**

The CVTDQ2PS instruction converts four packed 32-bit signed integer values in an XMM register or a 128-bit memory location to four packed single-precision floating-point values and writes the converted values in another XMM register. If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register.

Mnemonic	Opcode	Description
CVTDQ2PS <i>xmm1, xmm2/mem128</i>	0F 5B /r	Converts packed doubleword integer values in an XMM register or 128-bit memory location to packed single-precision floating-point values in the destination XMM register.

**Related Instructions**

CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

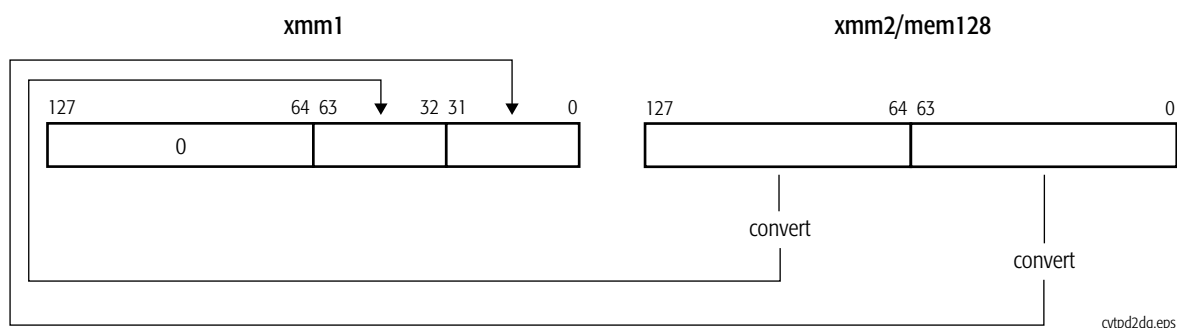
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPD2DQ**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers**

The CVTPD2DQ instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integers and writes the converted values in the low-order 64 bits of another XMM register. The high-order 64 bits in the destination XMM register are cleared to all 0s.

Mnemonic	Opcode	Description
CVTPD2DQ <i>xmm1, xmm2/mem128</i>	F2 0F E6	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integers in the destination XMM register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PD, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

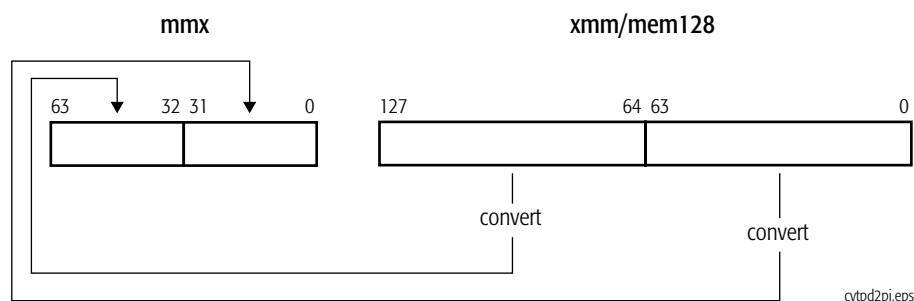
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPD2PI**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers**

The CVTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPD2PI <i>mmx, xmm2/mem128</i>	66 0F 2D /r	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integers values in the destination MMX register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ		RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

Exceptions

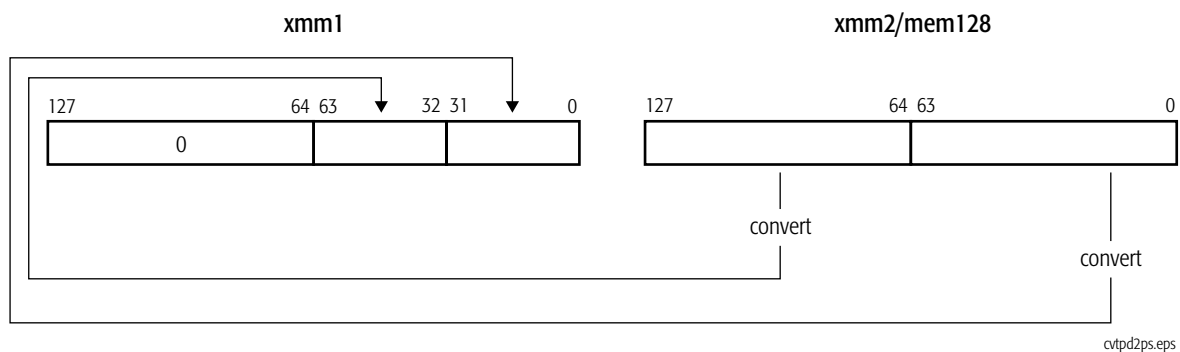
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPD2PS**Convert Packed Double-Precision Floating-Point to Packed Single-Precision Floating-Point**

The CVTPD2PS instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed single-precision floating-point values and writes the converted values in the low-order 64 bits of another XMM register. The high-order 64 bits in the destination XMM register are cleared to all 0s.

Mnemonic	Opcode	Description
CVTPD2PS <i>xmm1, xmm2/mem128</i>	66 0F 5A/ <i>r</i>	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed single-precision floating-point values in the destination XMM register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register.

Related Instructions

CVTPS2PD, CVTSD2SS, CVTSS2SD

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

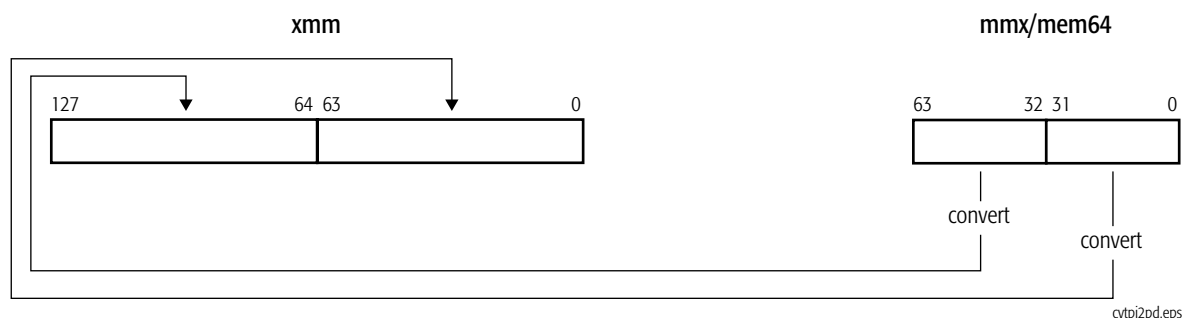
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPI2PD**Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point**

The CVTPI2PD instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two double-precision floating-point values and writes the converted values in an XMM register.

Mnemonic	Opcode	Description
CVTPI2PD <i>xmm, mmx/mem64</i>	66 0F 2A/r	Converts two packed doubleword integer values in an MMX register or 64-bit memory location to two packed double-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

None.

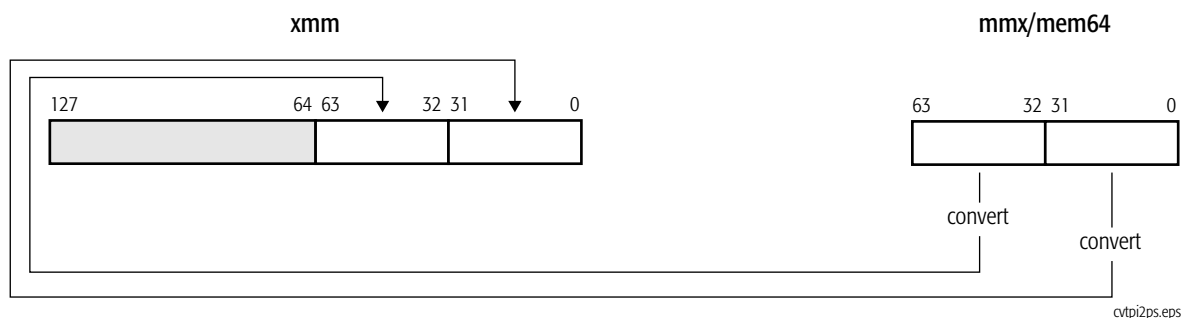
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPI2PS**Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point**

The CVTPI2PS instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two single-precision floating-point values and writes the converted values in the low-order 64 bits of an XMM register. The high-order 64 bits of the XMM register are not modified.

Mnemonic	Opcode	Description
CVTPI2PS <i>xmm, mmx/mem64</i>	0F 2A/ <i>r</i>	Converts packed doubleword integer values in an MMX register or 64-bit memory location to single-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

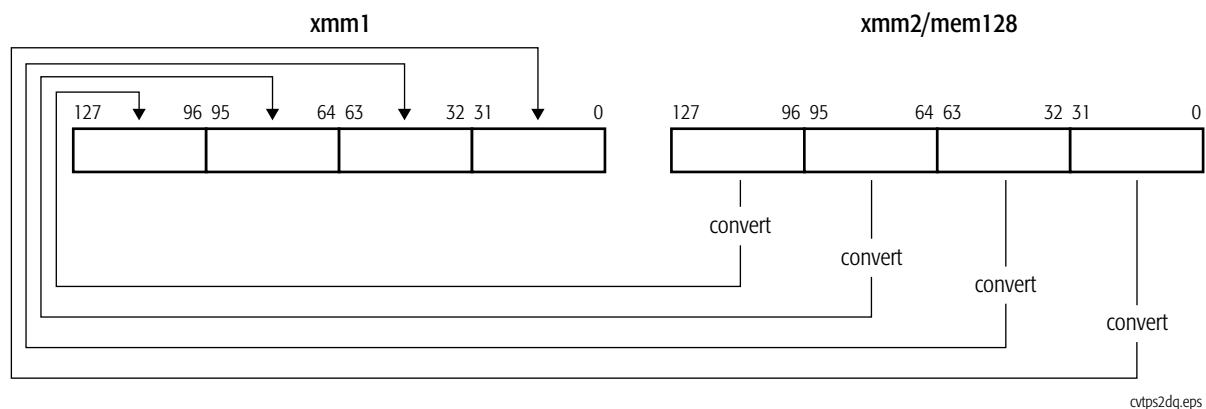
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPS2DQ**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers**

The CVTPS2DQ instruction converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed 32-bit signed integer values and writes the converted values in another XMM register.

Mnemonic	Opcode	Description
CVTPS2DQ <i>xmm1, xmm2/mem128</i>	66 0F 5B /r	Converts four packed single-precision floating-point values in an XMM register or 128-bit memory location to four packed doubleword integers in the destination XMM register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

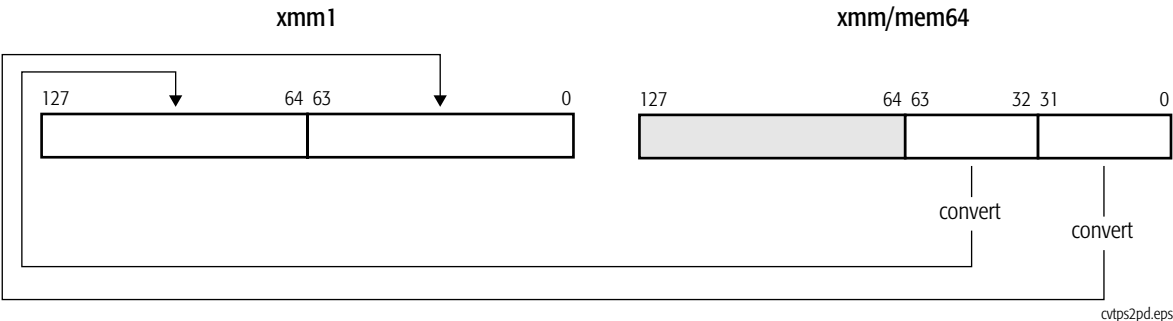
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSP2PD

Convert Packed Single-Precision Floating-Point
to Packed Double-Precision Floating-Point

The CVTSP2PD instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed double-precision floating-point values and writes the converted values in another XMM register.

Mnemonic	Opcode	Description
CVTSP2PD <i>xmm1, xmm2/mem64</i>	0F 5A/ <i>r</i>	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to packed double-precision floating-point values in the destination XMM register.



Related Instructions

CVTPD2PS, CVTSD2SS, CVTSS2SD

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

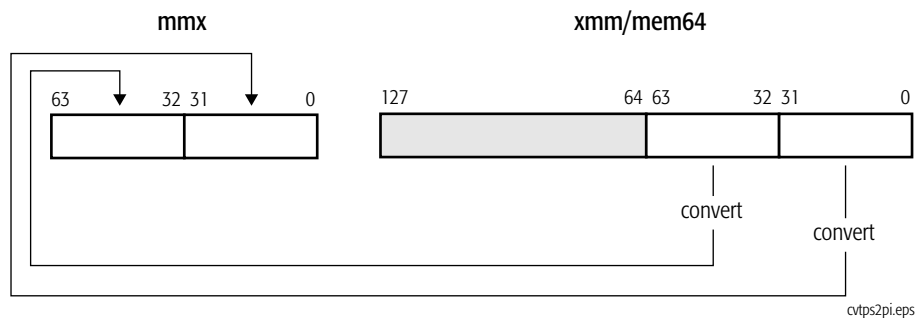
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CVTPS2PI**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers**

The CVTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integers and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPS2PI <i>mmx, xmm/mem64</i>	0F 2D /r	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to packed doubleword integers in the destination MMX register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word and the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

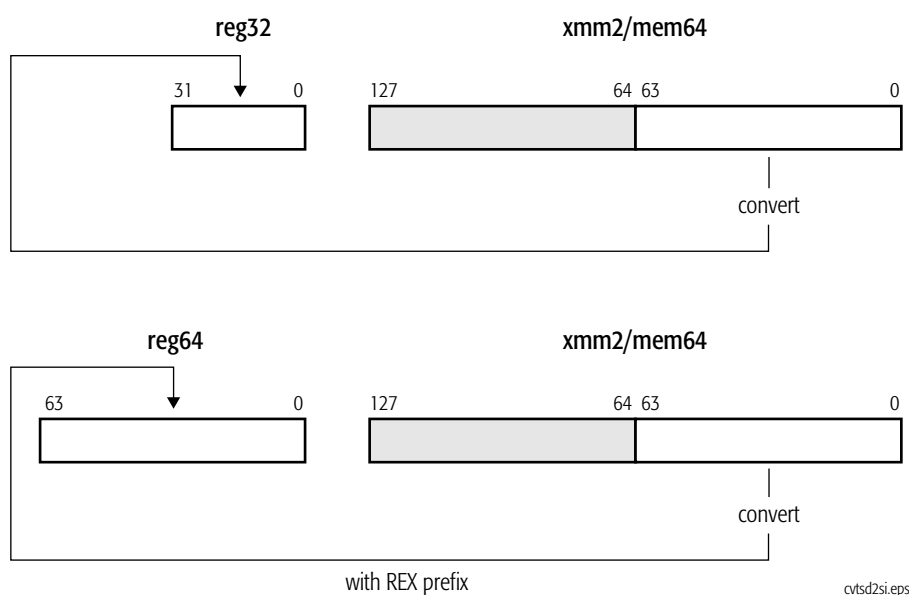
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSD2SI**Convert Scalar Double-Precision Floating-Point to Signed Doubleword or Quadword Integer**

The CVTSD2SI instruction converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit or 64-bit signed integer and writes the converted value in a general-purpose register.

Mnemonic	Opcode	Description
CVTSD2SI <i>reg32, xmm/mem64</i>	F2 0F 2D /r	Converts a packed double-precision floating-point value in an XMM register or 64-bit memory location to a doubleword integer in a general-purpose register.
CVTSD2SI <i>reg64, xmm/mem64</i>	F2 0F 2D /r	Converts a packed double-precision floating-point value in an XMM register or 64-bit memory location to a quadword integer in a general-purpose register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$) or quadword value (-2^{63} to $+2^{63}-1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ		RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

Exceptions

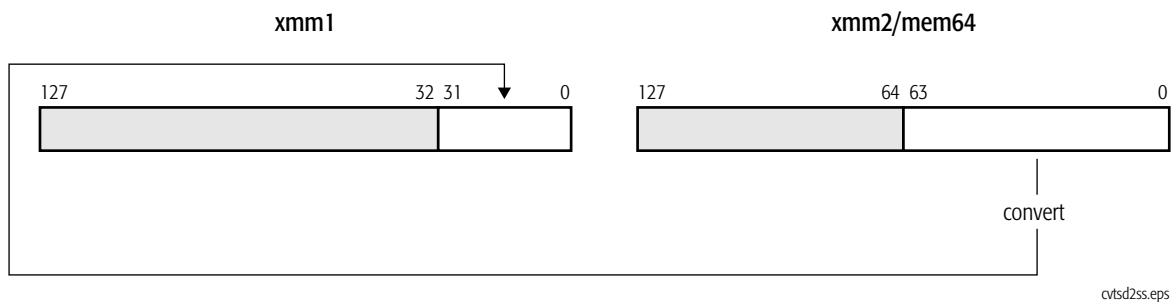
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSD2SS**Convert Scalar Double-Precision Floating-Point to Scalar Single-Precision Floating-Point**

The CVTSD2SS instruction converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a single-precision floating-point value and writes the converted value in the low-order 64 bits of another XMM register. The three high-order doublewords in the destination XMM register are not modified. If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register.

Mnemonic	Opcode	Description
CVTSD2SS <i>xmm1, xmm2/mem64</i>	F2 0F 5A/ <i>r</i>	Converts a scalar double-precision floating-point value in an XMM register or 64-bit memory location to a scalar single-precision floating-point value in the destination XMM register.

**Related Instructions**

CVTPD2PS, CVTPS2PD, CVTSS2SD

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

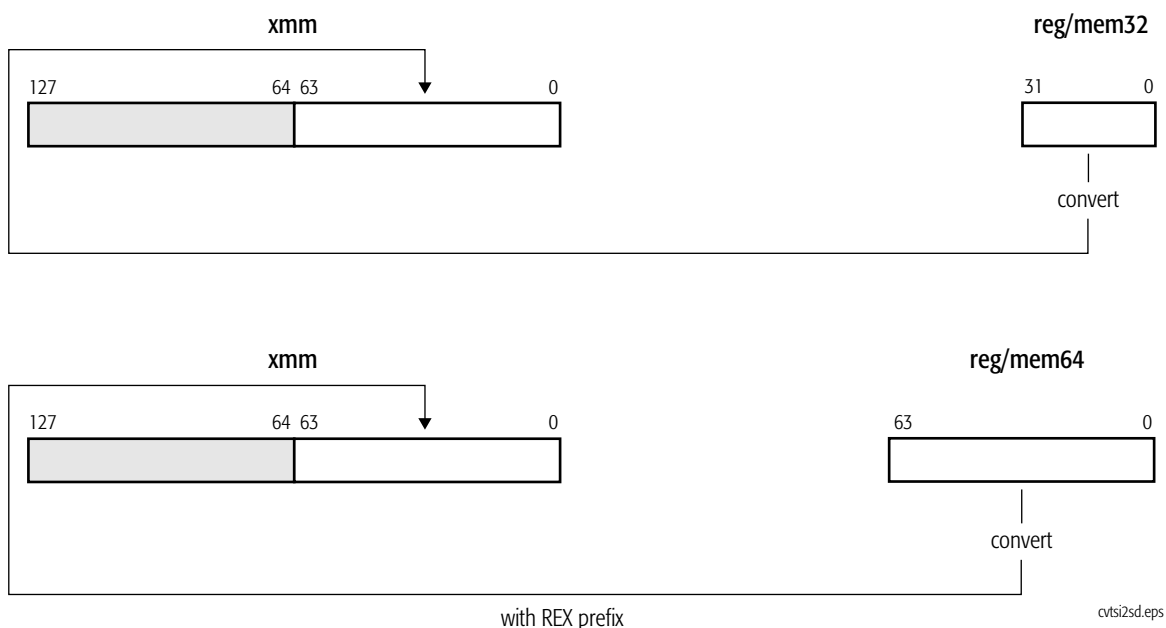
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSI2SD**Convert Signed Doubleword or Quadword Integer to Scalar Double-Precision Floating-Point**

The CVTSI2SD instruction converts a 32-bit or 64-bit signed integer value in a general-purpose register or memory location to a double-precision floating-point value and writes the converted value in the low-order 64 bits of an XMM register. The high-order 64 bits in the destination XMM register are not modified.

Mnemonic	Opcode	Description
CVTSI2SD <i>xmm, reg/mem32</i>	F2 0F 2A/r	Converts a doubleword integer in a general-purpose register or 32-bit memory location to a double-precision floating-point value in the destination XMM register.
CVTSI2SD <i>xmm, reg/mem64</i>	F2 0F 2A/r	Converts a quadword integer in a general-purpose register or 64-bit memory location to a double-precision floating-point value in the destination XMM register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

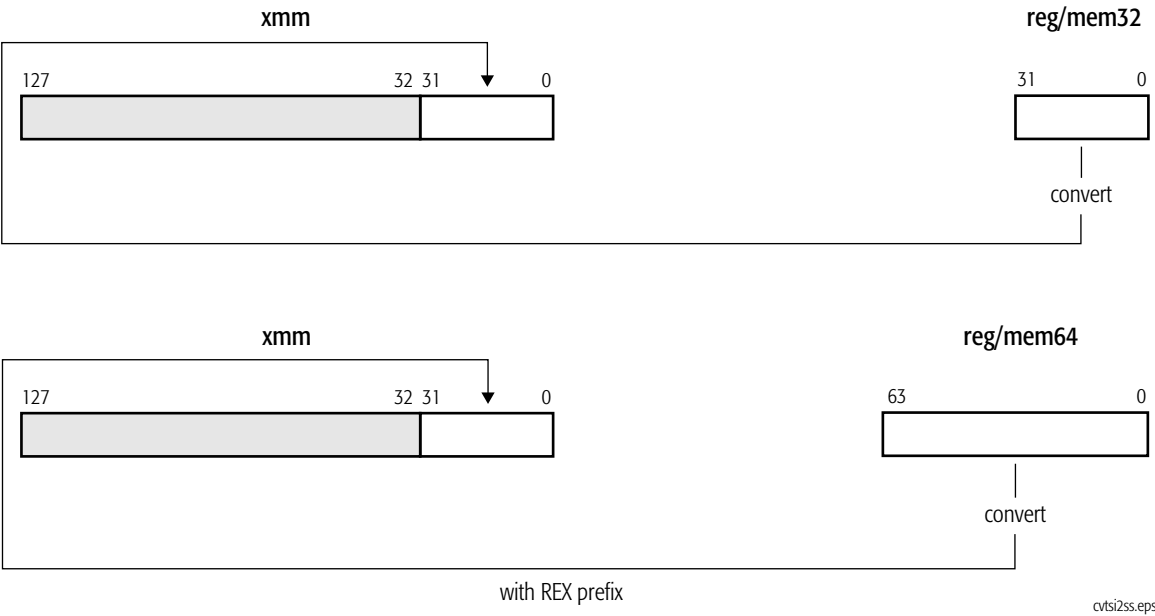
Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSI2SS

Convert Signed Doubleword or Quadword Integer to Scalar Single-Precision Floating-Point

The CVTSI2SS instruction converts a 32-bit or 64-bit signed integer value in a general-purpose register or memory location to a single-precision floating-point value and writes the converted value in the low-order 32 bits of an XMM register. The three high-order doublewords in the destination XMM register are not modified.

Mnemonic	Opcode	Description
CVTSI2SS <i>xmm, reg/mem32</i>	F3 0F 2A/r	Converts a doubleword integer in a general-purpose register or 32-bit memory location to a single-precision floating-point value in the destination XMM register.
CVTSI2SS <i>xmm, reg/mem64</i>	F3 0F 2A/r	Converts a quadword integer in a general-purpose register or 32-bit memory location to a single-precision floating-point value in the destination XMM register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE	
											1/0						
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																	

Exceptions

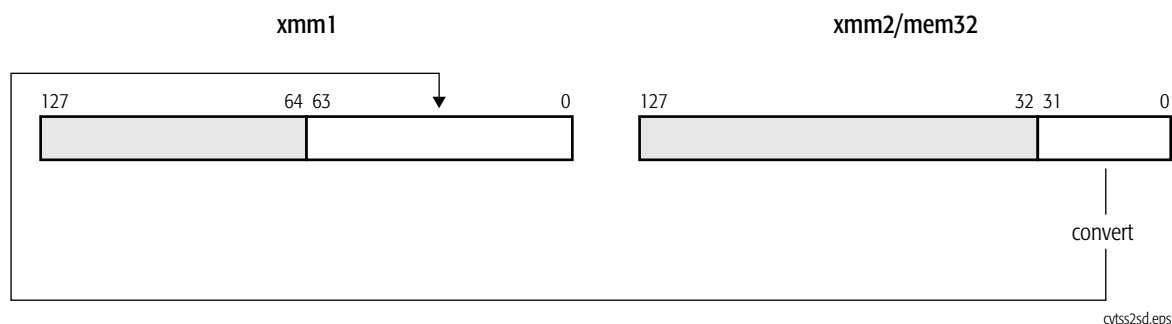
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTSS2SD**Convert Scalar Single-Precision Floating-Point to Scalar Double-Precision Floating-Point**

The CVTSS2SD instruction converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a double-precision floating-point value and writes the converted value in the low-order 64 bits of another XMM register. The high-order 64 bits in the destination XMM register are not modified.

Mnemonic	Opcode	Description
CVTSS2SD <i>xmm1, xmm2/mem32</i>	F3 0F 5A/ <i>r</i>	Converts scalar single-precision floating-point value in an XMM register or 32-bit memory location to double-precision floating-point value in the destination XMM register.

**Related Instructions**

CVTPD2PS, CVTPS2PD, CVTSD2SS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE	
															1/0	1/0	
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:

A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

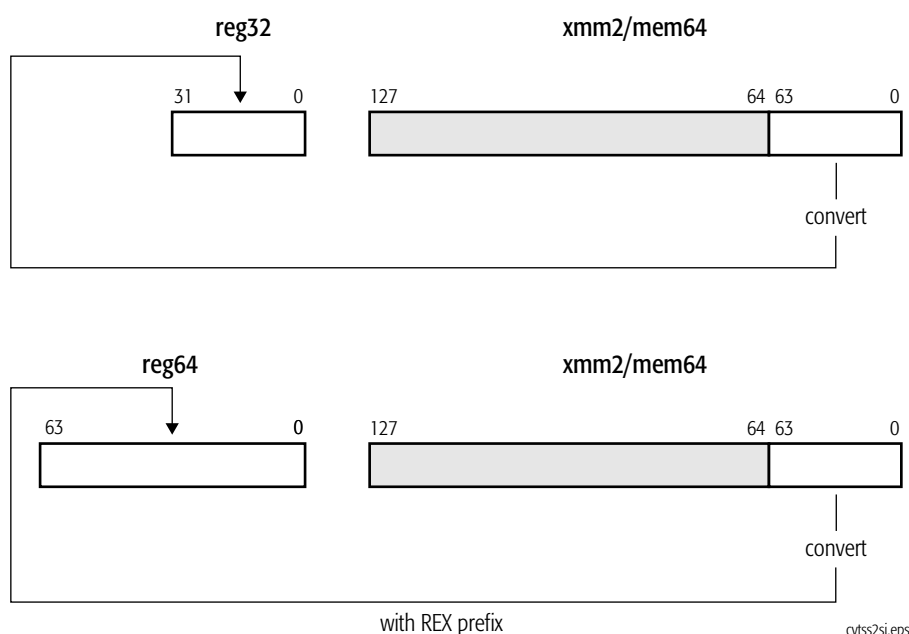
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

CVTSS2SI**Convert Scalar Single-Precision Floating-Point to Signed Doubleword or Quadword Integer**

The CVTSS2SI instruction converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit or 64-bit signed integer value and writes the converted value in a general-purpose register.

Mnemonic	Opcode	Description
<code>CVTSS2SI reg32, xmm/mem32</code>	F3 0F 2D /r	Converts a single-precision floating-point value in an XMM register or 32-bit memory location to a doubleword integer value in a general-purpose register.
<code>CVTSS2SI reg64, xmm/mem32</code>	F3 0F 2D /r	Converts a single-precision floating-point value in an XMM register or 32-bit memory location to a quadword integer value in a general-purpose register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$) or quadword value (-2^{63} to $+2^{63}-1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:

A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

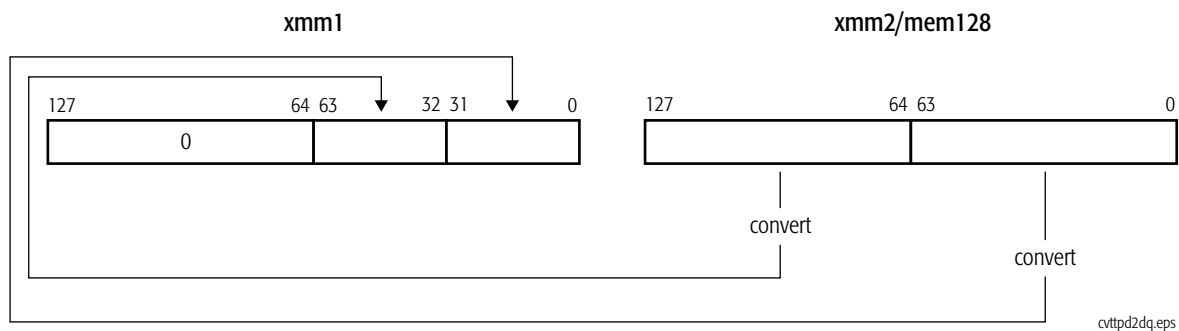
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPD2DQ**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPD2DQ instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in the low-order 64 bits of another XMM register. The high-order 64 bits of the destination XMM register are cleared to all 0s.

Mnemonic	Opcode	Description
CVTTPD2DQ <i>xmm1, xmm2/mem128</i>	66 0F E6	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integer values in the destination XMM register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

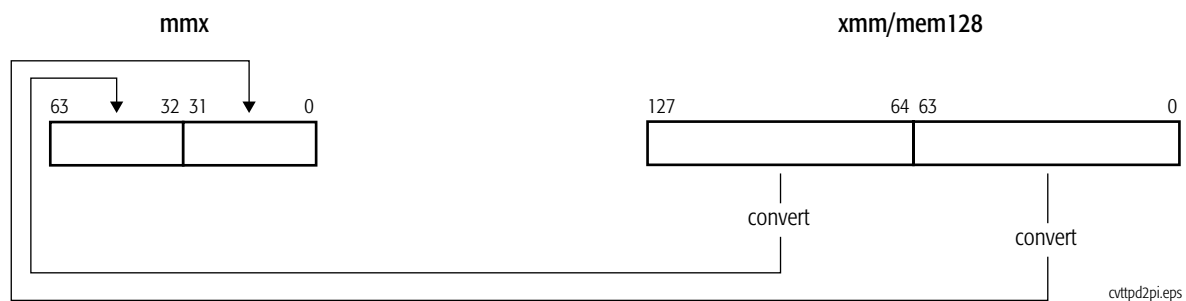
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPD2PI**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTTPD2PI <i>mmx, xmm/mem128</i>	66 0F 2C/ <i>r</i>	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integer values in the destination MMX register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

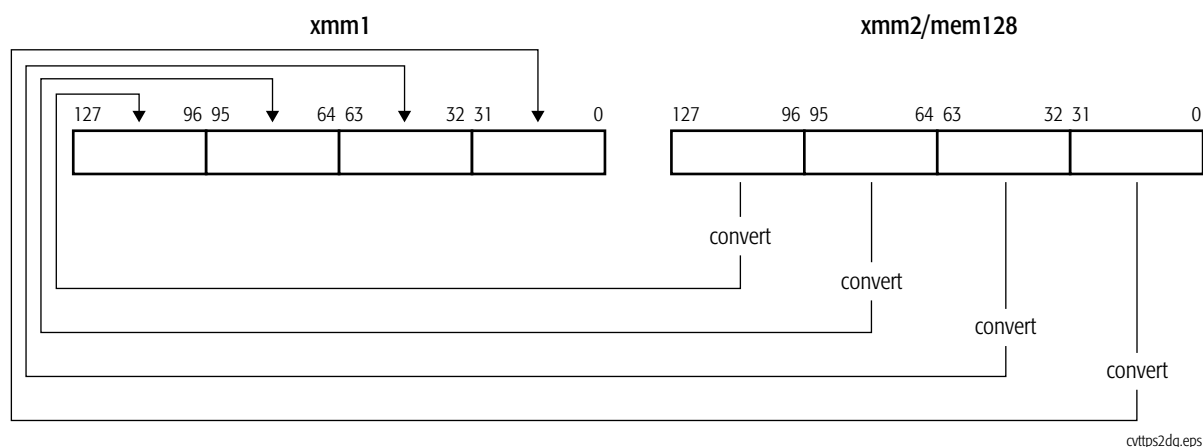
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPS2DQ**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPS2DQ instruction converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed 32-bit signed integers and writes the converted values in another XMM register.

Mnemonic	Opcode	Description
CVTTPS2DQ <i>xmm1, xmm2/mem128</i>	F3 0F 5B /r	Converts packed single-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integer values in the destination XMM register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTTPS2DQ, CVTTPS2PI, CVTSS2SI, CVTSS2SI, CVTTPS2PI, CVTTPS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

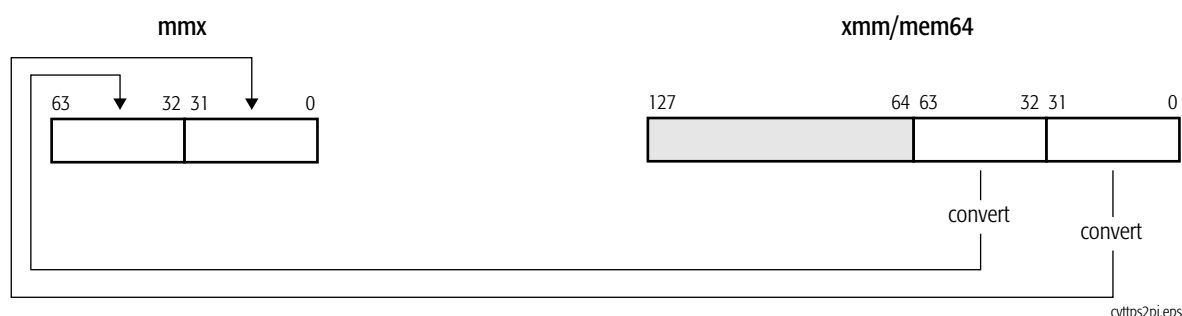
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPS2PI**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTTPS2PI <i>mmx xmm/mem64</i>	0F 2C/r	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to doubleword integer values in the destination MMX register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

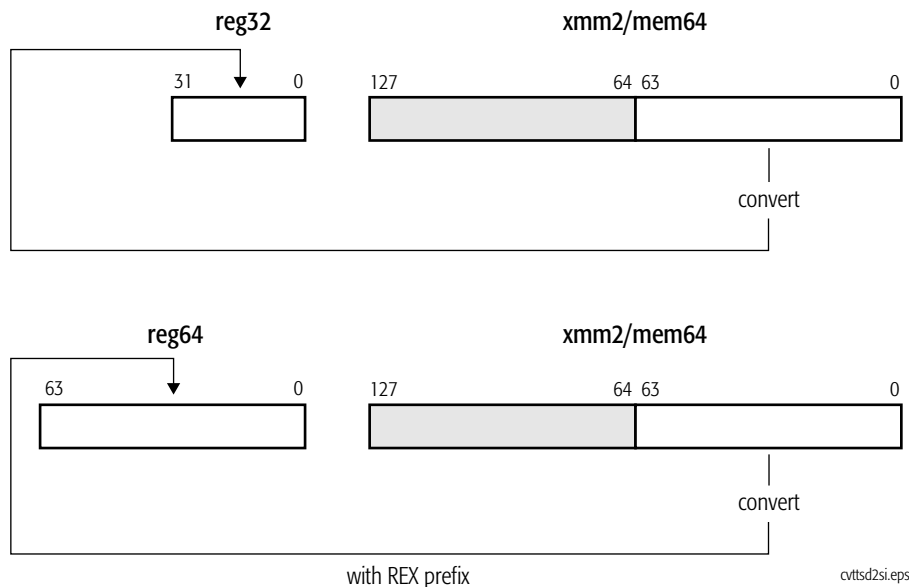
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTSD2SI**Convert Scalar Double-Precision Floating-Point to Signed Doubleword of Quadword Integer, Truncated**

The CVTTSD2SI instruction converts a double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit or 64-bit signed integer value and writes the converted value in a general-purpose register.

Mnemonic	Opcode	Description
CVTTSD2SI <i>reg32, xmm/mem64</i>	F2 0F 2C /r	Converts scalar double-precision floating-point value in an XMM register or 64-bit memory location to a doubleword signed integer value in a general-purpose register. Inexact results are truncated.
CVTTSD2SI <i>reg64, xmm/mem64</i>	F2 0F 2C /r	Converts scalar double-precision floating-point value in an XMM register or 64-bit memory location to a quadword signed integer value in a general-purpose register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$) or quadword value (-2^{63} to $+2^{63}-1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ		RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

Exceptions

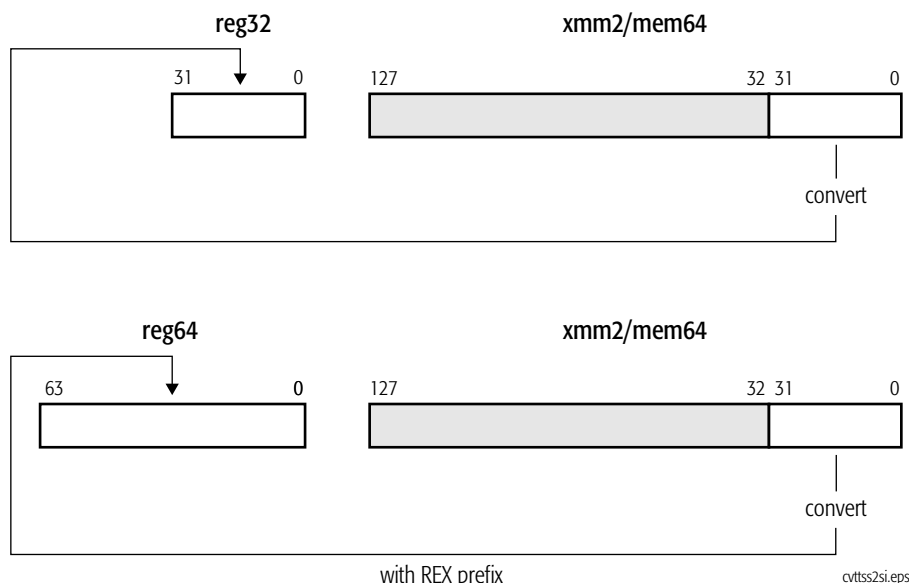
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTSS2SI**Convert Scalar Single-Precision Floating-Point to Signed Doubleword or Quadword Integer, Truncated**

The CVTTSS2SI instruction converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit or 64-bit signed integer value and writes the converted value in a general-purpose register.

Mnemonic	Opcode	Description
CVTTSS2SI <i>reg32, xmm/mem32</i>	F3 0F 2C /r	Converts scalar single-precision floating-point value in an XMM register or 32-bit memory location to a signed doubleword integer value in a general-purpose register. Inexact results are truncated.
CVTTSS2SI <i>reg64, xmm/mem32</i>	F3 0F 2C /r	Converts scalar single-precision floating-point value in an XMM register or 32-bit memory location to a signed quadword integer value in a general-purpose register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$) or quadword value (-2^{63} to $+2^{63}-1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

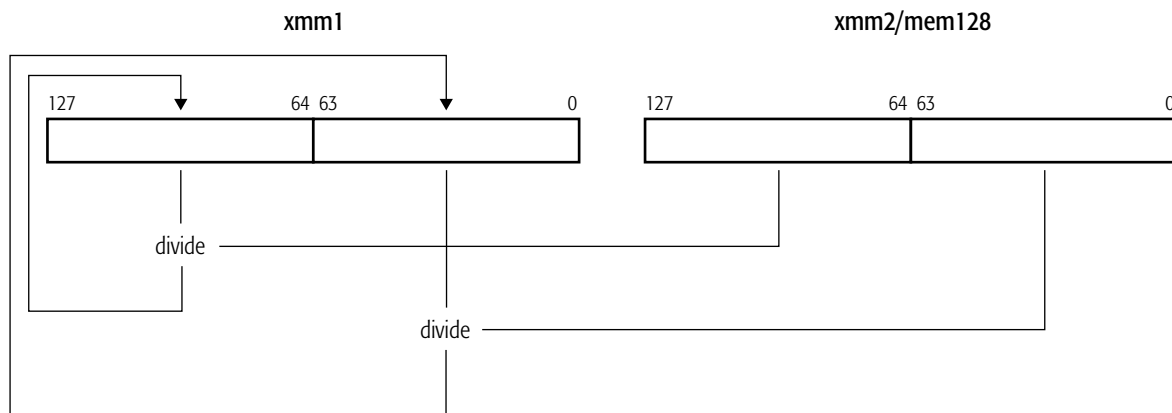
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

DIVPD**Divide Packed Double-Precision Floating-Point**

The DIVPD instruction divides each of the two packed double-precision floating-point values in the first source operand by the corresponding packed double-precision floating-point value in the second source operand and writes the result of each division in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
DIVPD <i>xmm1, xmm2/mem128</i>	66 0F 5E /r	Divides packed double-precision floating-point values in an XMM register by the packed double-precision floating-point values in another XMM register or 128-bit memory location.



divpd.eps

Related Instructions

DIVPS, DIVSD, DIVSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0	1/0	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

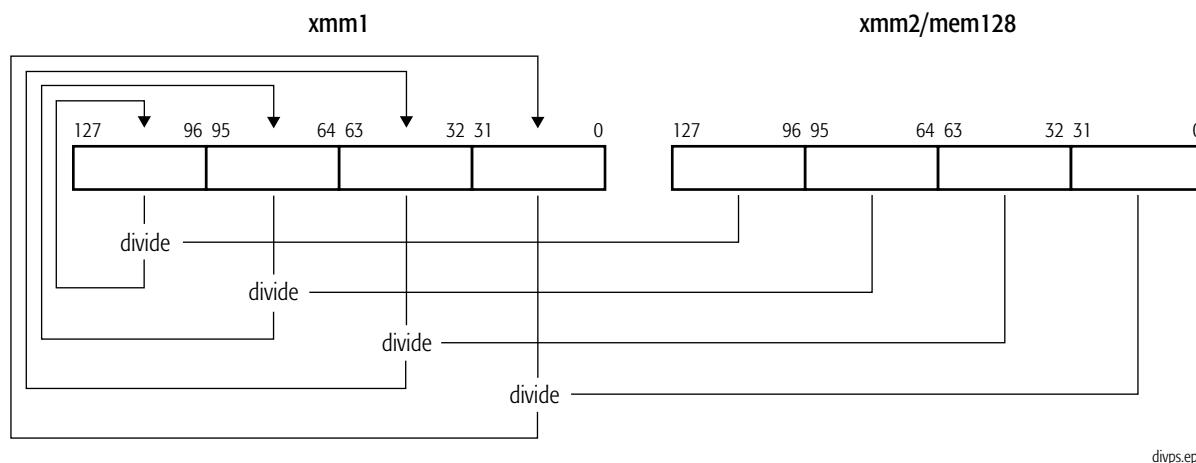
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	Attempt to divide non-zero operand by zero.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

DIVPS**Divide Packed Single-Precision Floating-Point**

The DIVPS instruction divides each of the four packed single-precision floating-point values in the first source operand by the corresponding packed single-precision floating-point value in the second source operand and writes the result of each division in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
DIVPS <i>xmm1, xmm2mem/128</i>	0F 5E/ <i>r</i>	Divides packed single-precision floating-point values in an XMM register by the packed single-precision floating-point values in another XMM register or 128-bit memory location.

**Related Instructions**

DIVPD, DIVSD, DIVSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0	1/0	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.

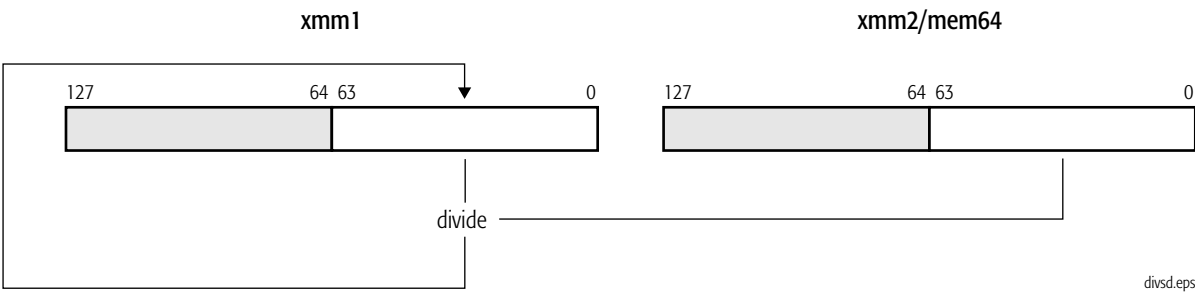
Exception	Real	Virtual 8086	Protected	Cause of Exception
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	Attempt to divide non-zero operand by zero.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

DIVSD

Divide Scalar Double-Precision Floating-Point

The DIVSD instruction divides the double-precision floating-point value in the low-order quadword of the first source operand by the double-precision floating-point value in the low-order quadword of the second source operand and writes the result in the low-order quadword of the destination (first source). The high-order quadword of the destination is not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
DIVSD <i>xmm1, xmm2/mem64</i>	F2 0F 5E/ <i>r</i>	Divides low-order double-precision floating-point value in an XMM register by the low-order double-precision floating-point value in another XMM register or in a 64-bit memory location.



Related Instructions

DIVPD, DIVPS, DIVSS

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0	1/0	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.																	

Exceptions

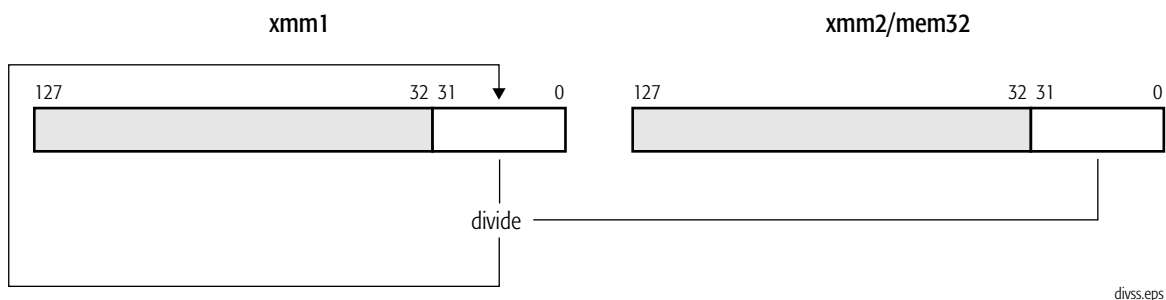
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).

Exception	Real	Virtual 8086	Protected	Cause of Exception
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	Attempt to divide non-zero operand by zero.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

DIVSS**Divide Scalar Single-Precision Floating-Point**

The DIVSS instruction divides the single-precision floating-point value in the low-order doubleword of the first source operand by the single-precision floating-point value in the low-order doubleword of the second source operand and writes the result in the low-order doubleword of the destination (first source). The three high-order doublewords of the destination are not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
DIVSS <i>xmm1, xmm2/mem32</i>	F3 0F 5E /r	Divides low-order single-precision floating-point value in an XMM register by the low-order single-precision floating-point value in another XMM register or in a 32-bit memory location.

**Related Instructions**

DIVPD, DIVPS, DIVSD

rFLAGS Affected

None.

MXCSR Flags Affected.

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0	1/0	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:

A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).

Exception	Real	Virtual 8086	Protected	Cause of Exception
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	Attempt to divide non-zero operand by zero.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FXRSTOR**Restore XMM, MMX, and x87 State**

The FXRSTOR instruction restores the XMM, MMX, and x87 state. The data loaded from memory is the state information previously saved using the FXSAVE instruction. Restoring data with FXRSTOR that had been previously saved with an FSAVE (rather than FXSAVE) instruction results in an incorrect restoration.

Mnemonic	Opcode	Description
FXRSTOR <i>mem512env</i>	OF AE /1	Restores XMM, MMX, and x87 state.

Unlike the FRSTOR instruction, FXRSTOR does not cause pending unmasked x87 floating-point exceptions in the restored image to be recognized when internal control of x87 exceptions is enabled (CR0.NE=1). When loading a new environment, use an FWAIT instruction after the FXRSTOR instruction to check for and handle any pending unmasked x87 exceptions.

If the restored MXCSR register contains a set bit in an exception status flag, and the corresponding exception mask bit is cleared (indicating an unmasked exception), loading the MXCSR register from memory does not cause a SIMD floating-point exception (#XM).

FXRSTOR executes faster than FRSTOR because it does not restore the x87 error pointers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXRSTOR, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXRSTOR instruction. If software running in 64-bit mode executes an FXRSTOR with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXRSTOR with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, the saved image of XMM0–XMM7 and MXCSR is not loaded into the processor. A general-protection exception occurs if there is an attempt to load a non-zero value to the bits in MXCSR that are defined as reserved and cleared (bits 31-16 and bit 6).

Related Instructions

FWAIT, FXSAVE

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

FXSAVE Save XMM, MMX, and x87 State

The FXSAVE instruction saves the XMM, MMX, and x87 state. A memory location that is not aligned on a 16-byte boundary causes a general-protection exception, or, in some cases, an alignment-check exception.

Mnemonic	Opcode	Description
FXSAVE <i>mem512env</i>	0F AE /0	Saves XMM, MMX, and x87 state in 512-byte memory location.

Unlike FSAVE and FNSAVE, FXSAVE does not alter the x87 tag bits. The contents of the saved MMX/x87 data registers are retained, thus indicating that the registers may be valid (or whatever other value the x87 tag bits indicated prior to the save). To invalidate the contents of the MMX/x87 data registers after FXSAVE, software must execute an FINIT instruction. Also, FXSAVE (like FNSAVE) does not check for pending unmasked x87 floating-point exceptions. An FWAIT instruction can be used for this purpose.

FXSAVE executes faster than FSAVE or FNSAVE because it does not save the x87 pointer registers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXSAVE, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXSAVE instruction. If software running in 64-bit mode executes an FXSAVE with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXSAVE with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, FXSAVE does not save the image of XMM0–XMM7 and MXCSR. For details about the CR4.OSFXSR bit, see “FXSAVE/FXRSTOR Support (OSFXSR) Bit” on page 63 of Volume 2.

Related Instructions

FINIT, FNSAVE, FRSTOR, FSAVE, FXRSTOR, LDMXCSR, STMXCSR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

LDMXCSR**Load MXCSR Control/Status Register**

The LDMXCSR instruction loads the MXCSR register with a 32-bit value from memory. The least-significant bit of the memory location is loaded in bit 0 of MXCSR. Bits 31-16 and bit 6 of the MXCSR are reserved and must be zero. A general-protection exception occurs if the LDMXCSR instruction attempts to load non-zero values into MXCSR bits 31-16 or bit 6.

The MXCSR register is described in Section 4.3, “Registers”, on page 143 of Volume 1.

Mnemonic	Opcode	Description
LDMXCSR <i>mem32</i>	0F AE/2	Loads MXCSR register with 32-bit value in memory.

Related Instructions

STMXCSR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

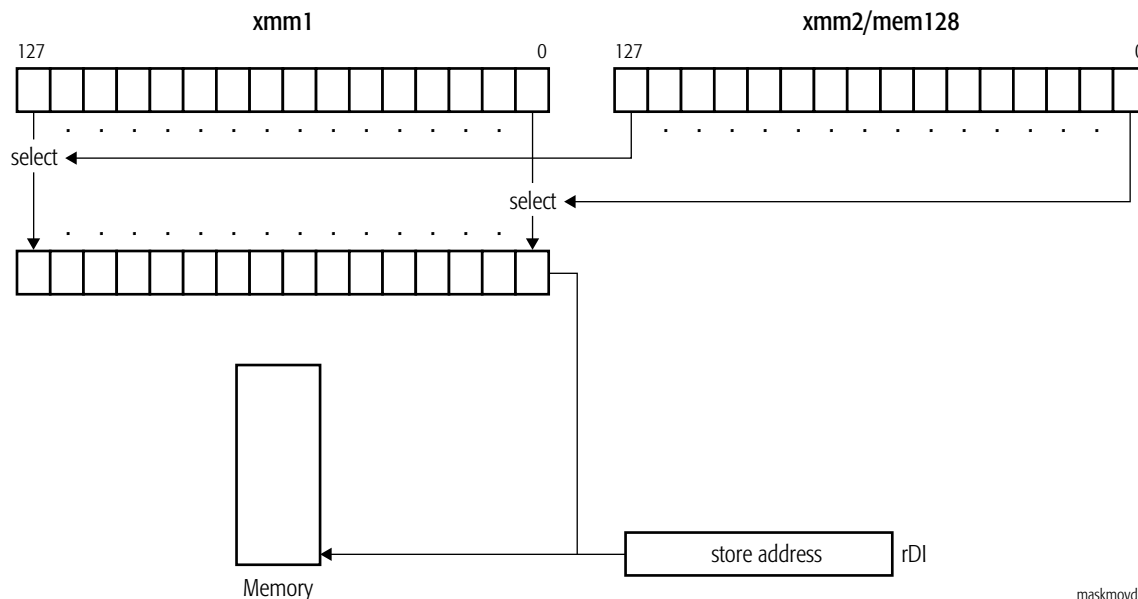
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MASKMOVDQU Masked Move Double Quadword Unaligned

The MASKMOVDQU instruction stores bytes from the first source operand as selected by the mask value in the second source operand (0 = no write and 1 = write) to a memory location specified in the rDI and DS registers. The first source operand is an XMM register, and the second source operand is another XMM register. The store address may be unaligned.

Mnemonic	Opcode	Description
MASKMOVDQU <i>xmm1, xmm2</i>	66 0F F7 /r	Store bytes from an XMM register selected by a mask value in another XMM register to a memory location.



maskmovdqu.eps

A mask value of all 0s results in the following behavior:

- No data is written to memory.
- Code and data breakpoints are not guaranteed to be signalled in all implementations.
- Exceptions associated with memory addressing and page faults are not guaranteed to be signalled in all implementations.
- The protection features of memory regions mapped as UC or WP are not guaranteed to be enforced in all implementations.

MASKMOVDQU implicitly uses weakly-ordered, write-combining buffering for the data, as described in Section 7.4, “Buffering and Combining Memory Writes”, on page 215 of Volume 2. For data that is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (refer to Section 6.6, “Cache and TLB Management”, on page 201 of Volume 2).

Related Instructions

MASKMOVQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

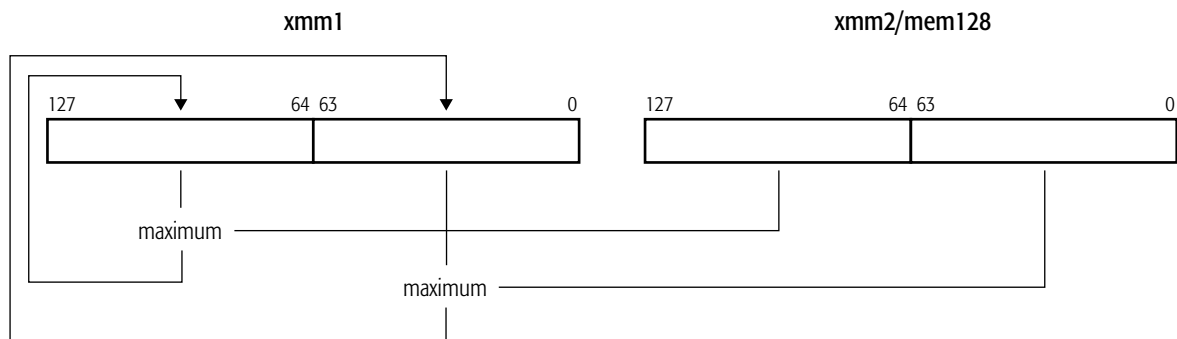
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MAXPD**Maximum Packed Double-Precision Floating-Point**

The MAXPD instruction compares each of the two packed double-precision floating-point values in the first source operand with the corresponding packed double-precision floating-point value in the second source operand and writes the maximum of the two values for each comparison in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MAXPD <i>xmm1, xmm2/mem128</i>	66 0F 5F /r	Compares two pairs of packed double-precision values in an XMM register and another XMM register or 128-bit memory location and writes the maximum value of each comparison in the destination XMM register.



maxpd.eps

If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPS, MAXSD, MAXSS, MINPD, MINPS, MINSR, MINSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

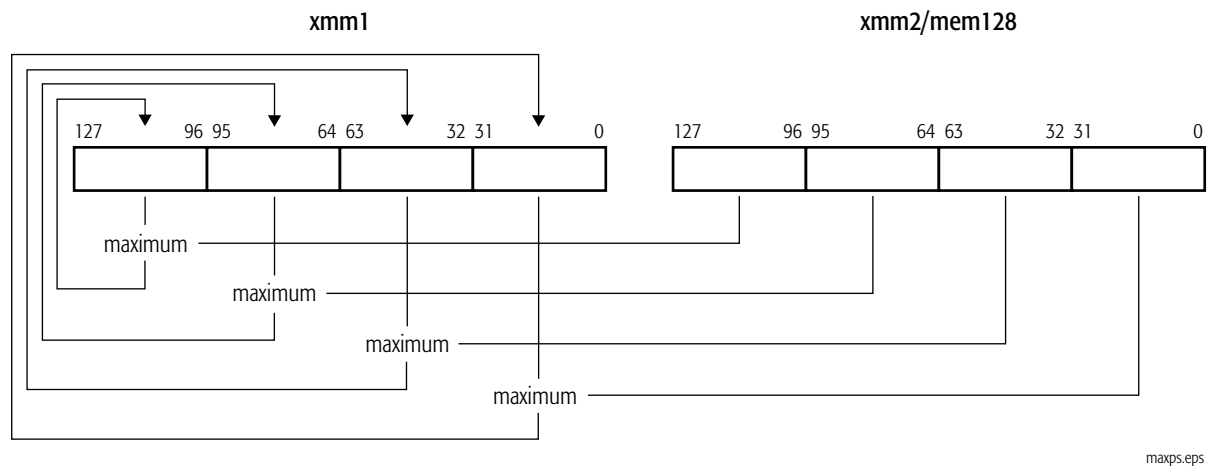
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MAXPS**Maximum Packed Single-Precision Floating-Point**

The MAXPS instruction compares each of the four packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the maximum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MAXPS <i>xmm1, xmm2/mem128</i>	OF 5F /r	Compares four pairs of packed single-precision values in an XMM register and another XMM register or 128-bit memory location and writes the maximum value of each comparison in the destination XMM register.



maxps.eps

If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXSD, MAXSS, MINPD, MINPS, MINSD, MINSS

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

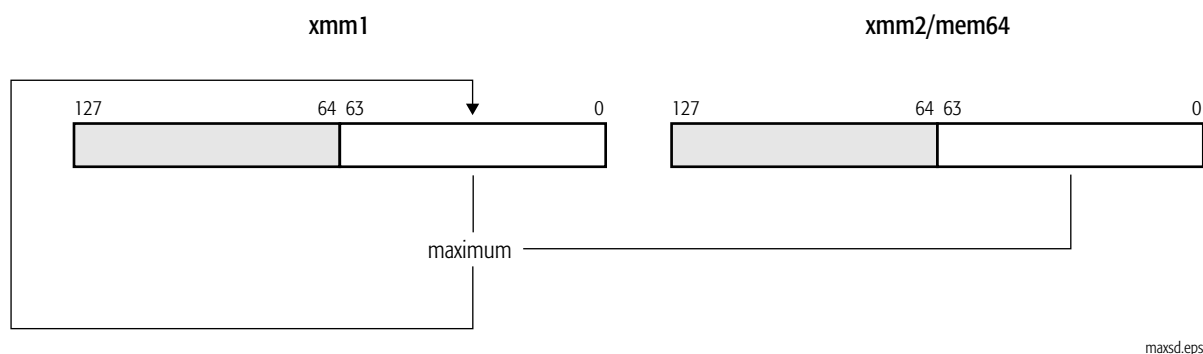
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MAXSD Maximum Scalar Double-Precision Floating-Point

The MAXSD instruction compares the double-precision floating-point value in the low-order 64 bits of the first source operand with the double-precision floating-point value in the low-order 64 bits of the second source operand and writes the maximum of the two values in the low-order quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 64-bit memory location. The high-order quadword of the destination XMM register is not modified.

Mnemonic	Opcode	Description
MAXSD <i>xmm1, xmm2/mem64</i>	F2 0F 5F /r	Compares scalar double-precision values in an XMM register and another XMM register or 64-bit memory location and writes the maximum of the two values in the destination XMM register.



If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSS, MINPD, MINPS, MINSR, MINSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

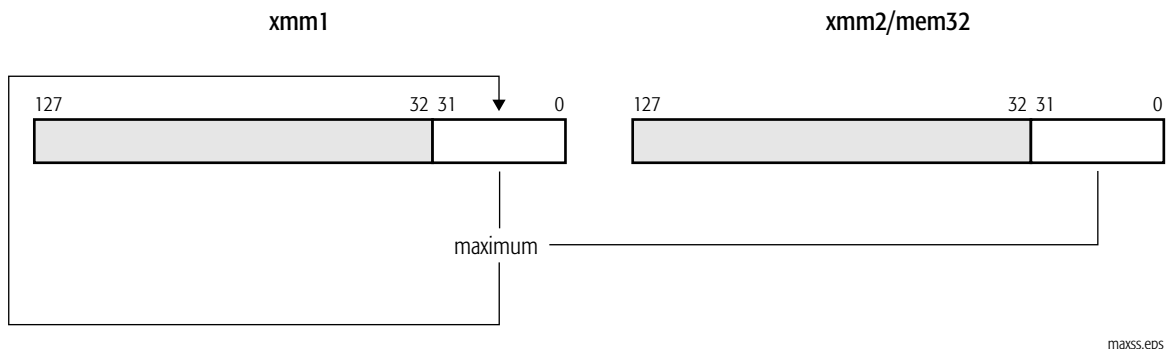
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MAXSS**Maximum Scalar Single-Precision Floating-Point**

The MAXSS instruction compares the single-precision floating-point value in the low-order 32 bits of the first source operand with the single-precision floating-point value in the low-order 32 bits of the second source operand and writes the maximum of the two values in the low-order 32 bits of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 32-bit memory location. The three high-order doublewords of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MAXSS <i>xmm1, xmm2/mem32</i>	F3 0F 5F /r	Compares scalar single-precision floating-point values in an XMM register and another XMM register or 32-bit memory location and writes the maximum of the two values in the destination XMM register.



maxss.eps

If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSD, MINPD, MINPS, MINSD, MINSS, PFMAX

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

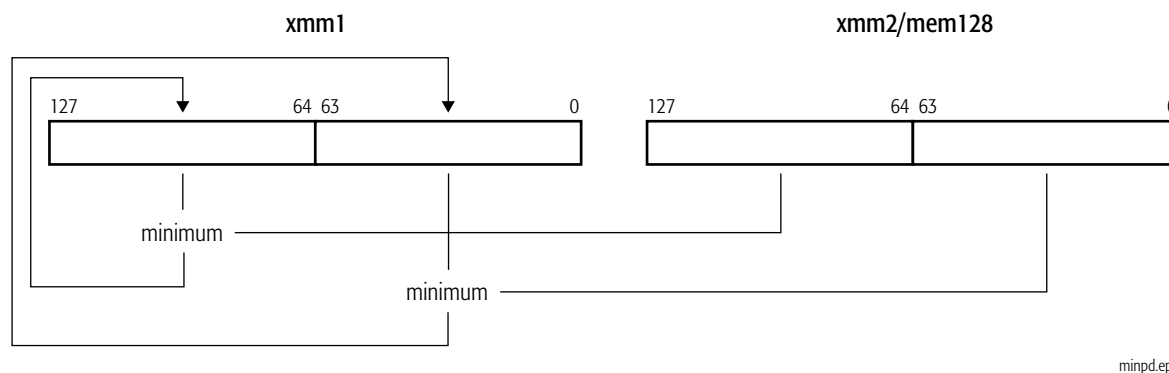
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MINPD**Minimum Packed Double-Precision Floating-Point**

The MINPD instruction compares each of the two packed double-precision floating-point values in the first source operand with the corresponding packed double-precision floating-point value in the second source operand and writes the minimum of the two values for each comparison in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 128-bit memory location.

Mnemonic	Opcode	Description
MINPD <i>xmm1, xmm2/mem128</i>	66 0F 5D /r	Compares two pairs of packed double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the minimum value of each comparison in the destination XMM register.



If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSD, MAXSS, MINPS, MINSD, MINSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

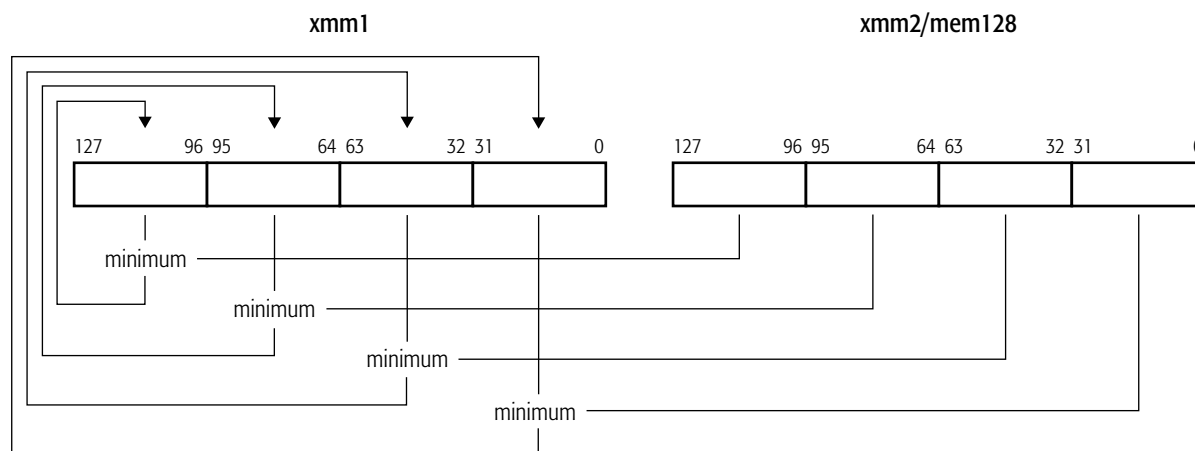
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MINPS**Minimum Packed Single-Precision Floating-Point**

The MINPS instruction compares each of the four packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the minimum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 128-bit memory location.

Mnemonic	Opcode	Description
MINPS <i>xmm1, xmm2/mem128</i>	0F 5D /r	Compares four pairs of packed single-precision values in an XMM register and another XMM register or 128-bit memory location and writes the minimum value of each comparison in the destination XMM register.



If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSD, MAXSS, MINPD, MINSD, MINSS, PFMIN

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.																	

Exceptions

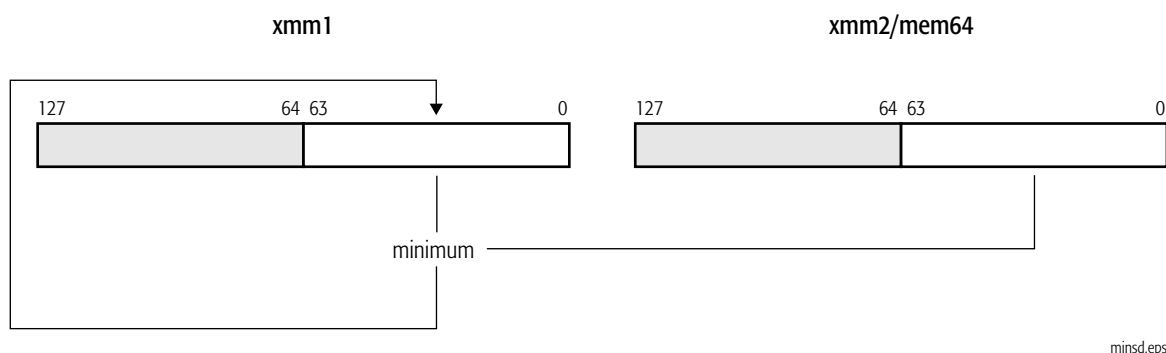
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MINSD**Minimum Scalar Double-Precision Floating-Point**

The MINSD instruction compares the double-precision floating-point value in the low-order 64 bits of the first source operand with the double-precision floating-point value in the low-order 64 bits of the second source operand and writes the minimum of the two values in the low-order 64 bits of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 64-bit memory location. The high-order quadword of the destination XMM register is not modified.

Mnemonic	Opcode	Description
MINSD <i>xmm1, xmm2/mem64</i>	F2 0F 5D /r	Compares scalar double-precision floating-point values in an XMM register and another XMM register or 64-bit memory location and writes the minimum of the two values in the destination XMM register.



If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSD, MAXSS, MINPD, MINPS, MINSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

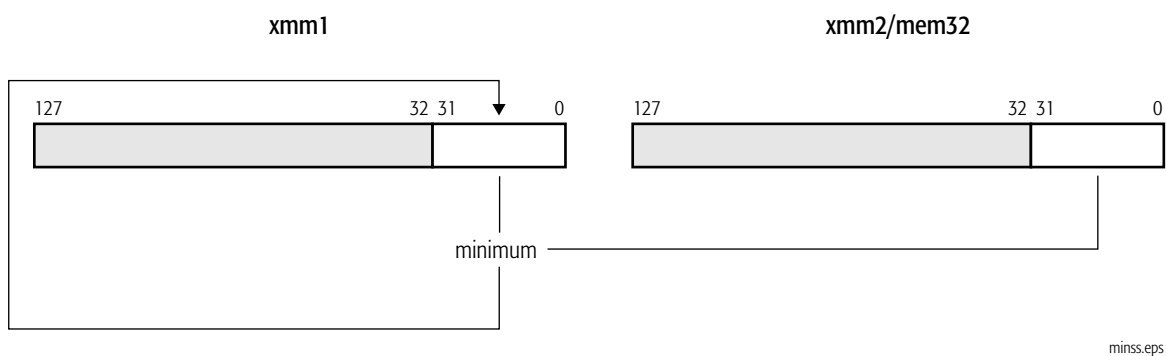
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MINSS**Minimum Scalar Single-Precision Floating-Point**

The MINSD instruction compares the single-precision floating-point value in the low-order 32 bits of the first source operand with the single-precision floating-point value in the low-order 32 bits of the second source operand and writes the maximum of the two values in the low-order 32 bits of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or a 32-bit memory location. The three high-order doublewords of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MINSS <i>xmm1, xmm2/mem32</i>	F3 0F 5D /r	Compares scalar single-precision floating-point values in an XMM register and another XMM register or 32-bit memory location and writes the minimum of the two values in the destination XMM register.



If both source operands are equal to zero, the value in the second source operand is returned. If either operand is a NaN (SNaN or QNaN), the second source operand is written to the destination.

Related Instructions

MAXPD, MAXPS, MAXSD, MAXSS, MINPD, MINPS, MINSD

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

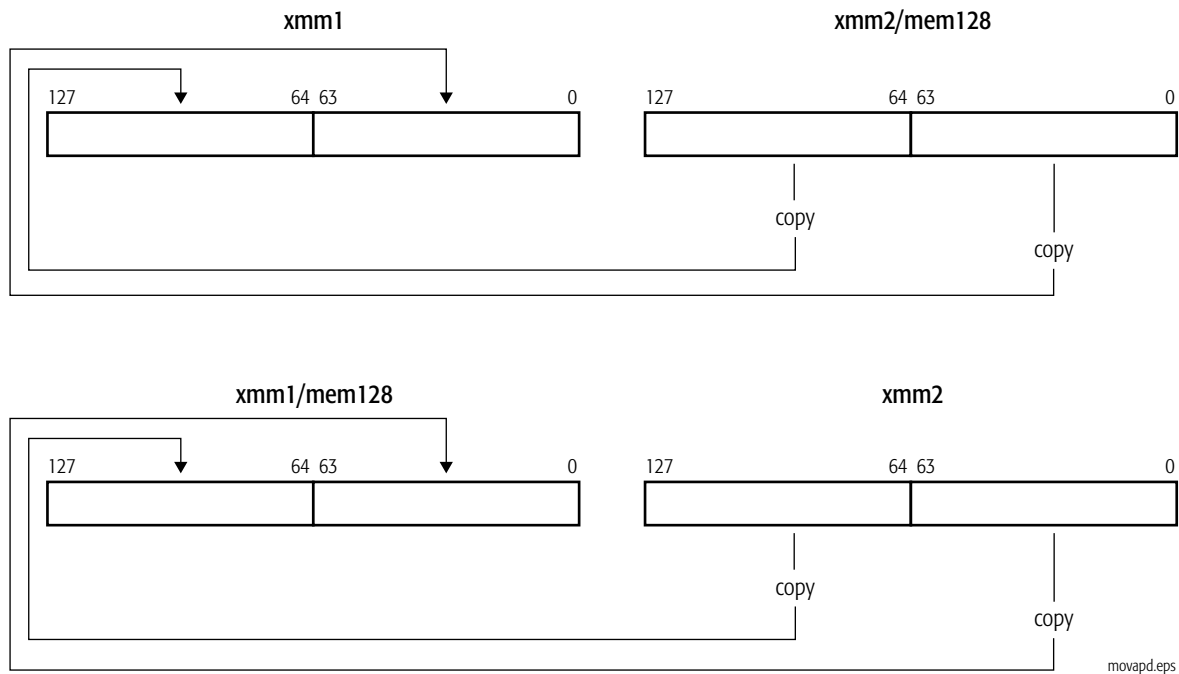
Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN or QNaN value.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

MOVAPD**Move Aligned Packed Double-Precision Floating-Point**

The MOVAPD instruction moves two packed double-precision floating-point values:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MOVAPD <i>xmm1, xmm2/mem128</i>	66 0F 28 /r	Moves packed double-precision floating-point value from an XMM register or 128-bit memory location to an XMM register.
MOVAPD <i>xmm1/mem128, xmm2</i>	66 0F 29 /r	Moves packed double-precision floating-point value from an XMM register to an XMM register or 128-bit memory location.



A memory operand that is not aligned on a 16-byte boundary causes a general-protection exception.

Related Instructions

MOVHPD, MOVLPD, MOVMSKPD, MOVSD, MOVUPD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

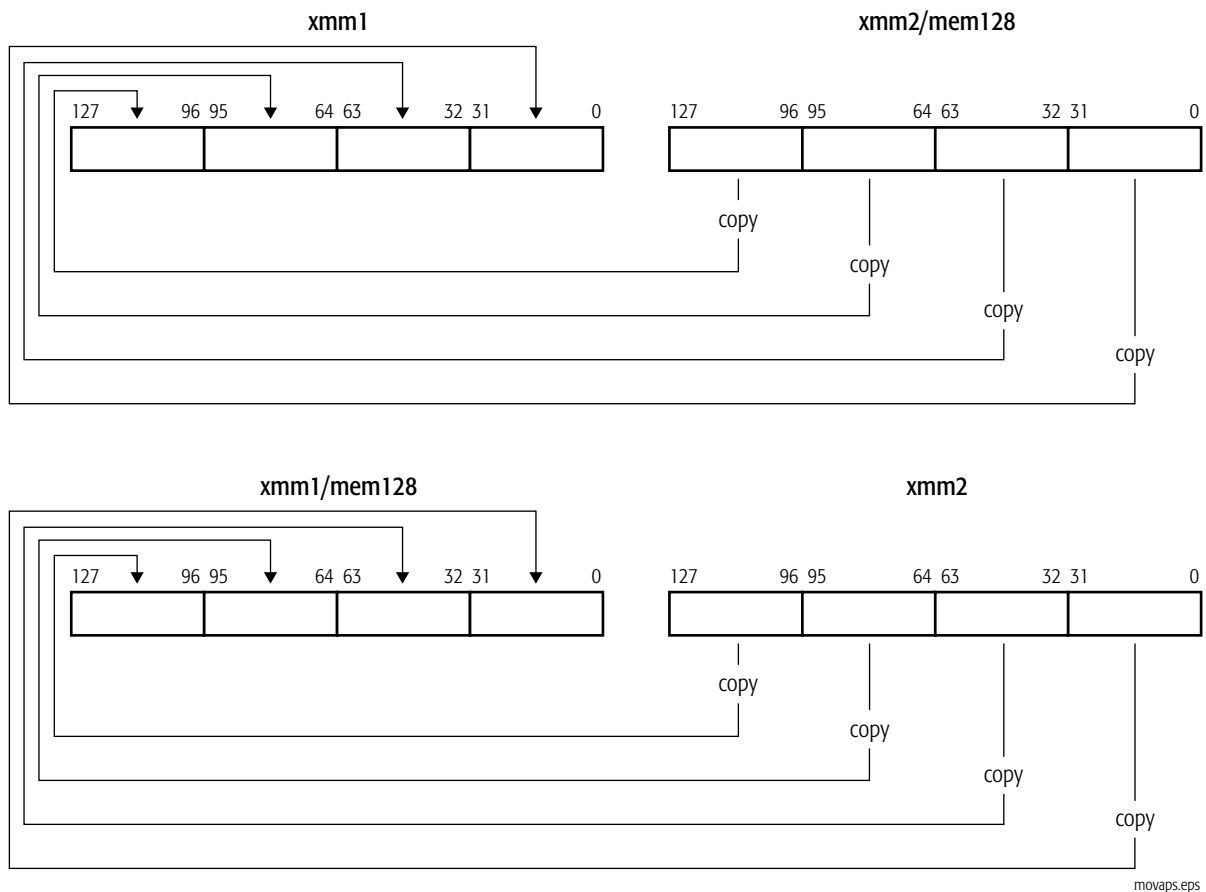
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVAPS**Move Aligned Packed Single-Precision Floating-Point**

The MOVAPS instruction moves four packed single-precision floating-point values:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MOVAPS <i>xmm1</i> , <i>xmm2/mem128</i>	0F 28 /r	Moves aligned packed single-precision floating-point value from an XMM register or 128-bit memory location to the destination XMM register.
MOVAPS <i>xmm1/mem128</i> , <i>xmm2</i>	0F 29 /r	Moves aligned packed single-precision floating-point value from an XMM register to the destination XMM register or 128-bit memory location.



movaps.eps

A memory operand that is not aligned on a 16-byte boundary causes a general-protection exception.

Related Instructions

MOVHLPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVSS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

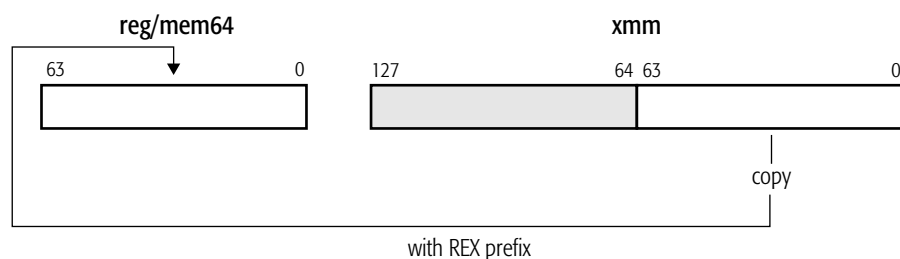
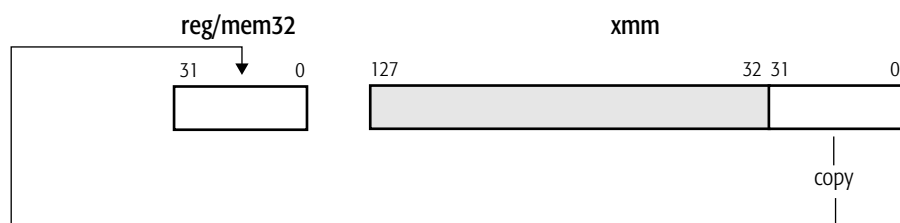
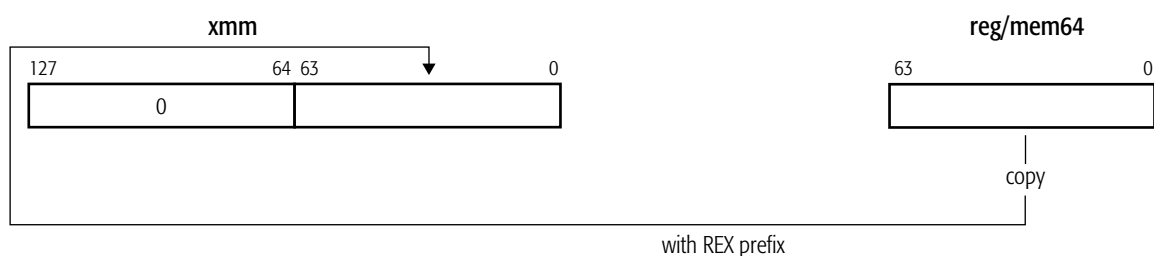
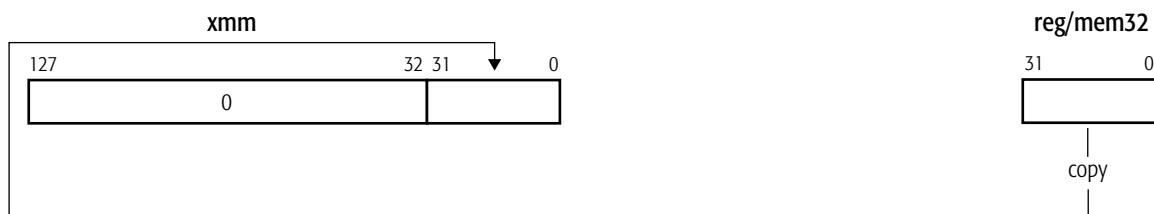
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVD**Move Doubleword or Quadword**

The MOVD instruction moves a 32-bit or 64-bit value:

- from a 32-bit or 64-bit general-purpose register or memory location to the low-order 32 or 64 bits of an XMM register, with zero-extension to 128 bits, or
- from the low-order 32 or 64 bits of an XMM to a 32-bit or 64-bit general-purpose register or memory location, or

Mnemonic	Opcode	Description
MOVD <i>xmm, reg/mem32</i>	66 0F 6E /r	Moves 32-bit value from a general-purpose register or 32-bit memory location to an XMM register.
MOVD <i>xmm, reg/mem64</i>	66 0F 6E /r	Moves 64-bit value from a general-purpose register or 64-bit memory location to an XMM register.
MOVD <i>reg/mem32, xmm</i>	66 0F 7E /r	Moves 32-bit value from an XMM register to a general-purpose register or 32-bit memory location.
MOVD <i>reg/mem64, xmm</i>	66 0F 7E /r	Moves 64-bit value from an XMM register to a general-purpose register or 64-bit memory location.



movd-128.eps

Related Instructions

MOVDQA, MOVDQU, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

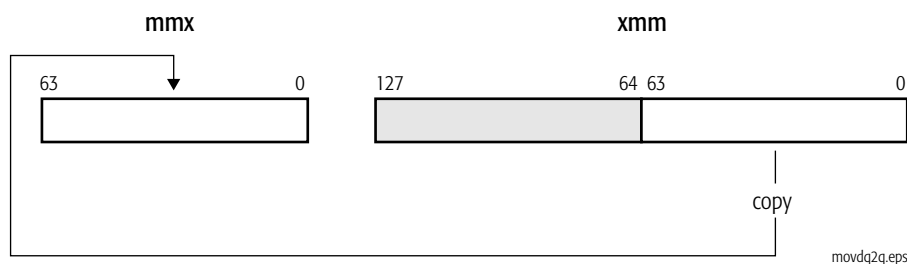
Exceptions

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVDQ2Q**Move Quadword to Quadword**

The MOVDQ2Q instruction moves the low-order 64-bit value in an XMM register to a 64-bit MMX register.

Mnemonic	Opcode	Description
MOVDQ2Q <i>mmx, xmm</i>	F2 0F D6	Moves low-order 64-bit value from an XMM register to the destination MMX register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

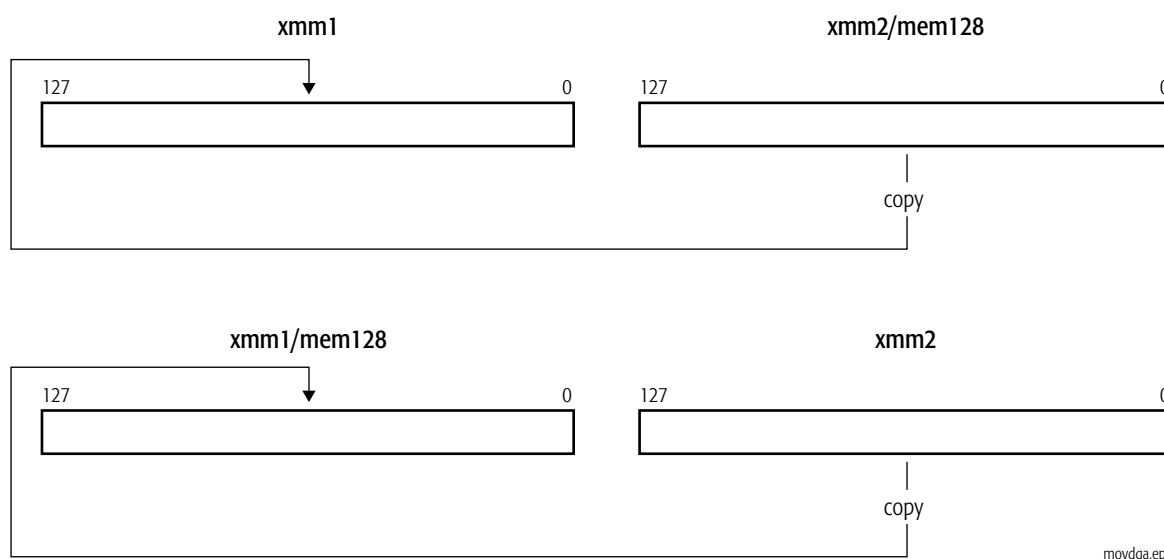
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

MOVDQA Move Aligned Double Quadword

The MOVDQA instruction moves an aligned 128-bit (double quadword) value:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to a 128-bit memory location or another XMM register.

Mnemonic	Opcode	Description
MOVDQA <i>xmm1</i> , <i>xmm2/mem128</i>	66 0F 6F /r	Moves 128-bit value from an XMM register or 128-bit memory location to the destination XMM register.
MOVDQA <i>xmm1/mem128</i> , <i>xmm2</i>	66 0F 7F /r	Moves 128-bit value from an XMM register to the destination XMM register or 128-bit memory location.



movdqa.eps

A memory operand that is not aligned on a 16-byte boundary causes a general-protection exception.

Related Instructions

MOVD, MOVDQU, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

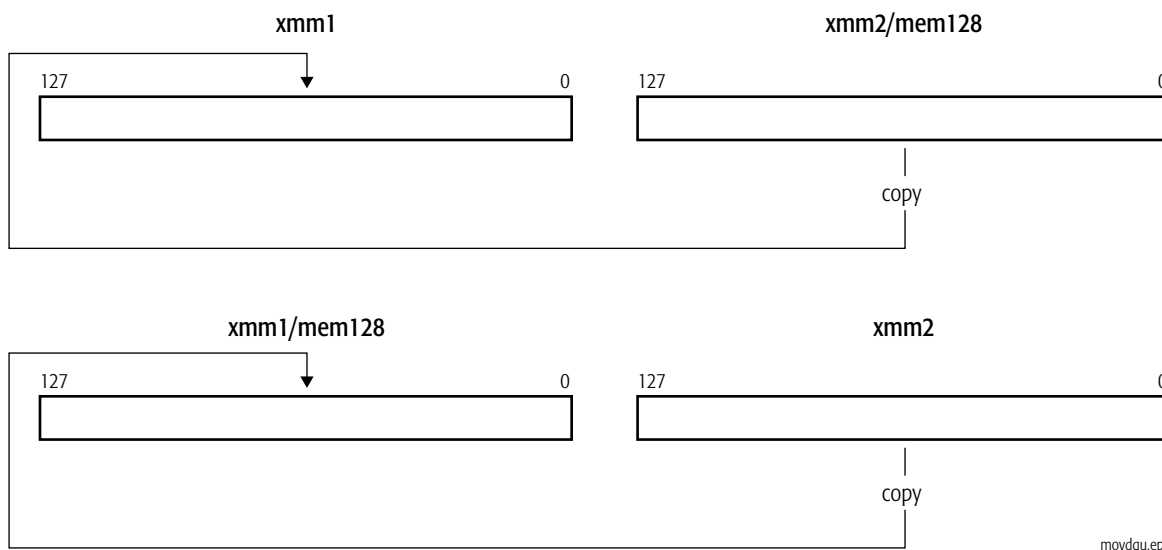
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVDQU Move Unaligned Double Quadword

The MOVDQU instruction moves an unaligned 128-bit (double quadword) value:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MOVDQU <i>xmm1</i> , <i>xmm2/mem128</i>	F3 0F 6F /r	Moves unaligned 128-bit value from an XMM register or 128-bit memory location to the destination XMM register.
MOVDQU <i>xmm1/mem128</i> , <i>xmm2</i>	F3 0F 7F /r	Moves unaligned 128-bit value from an XMM register to the destination XMM register or 128-bit memory location.



movdqu.eps

Memory operands that are not aligned on a 16-byte boundary do not cause a general-protection exception.

Related Instructions

MOVD, MOVDQA, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

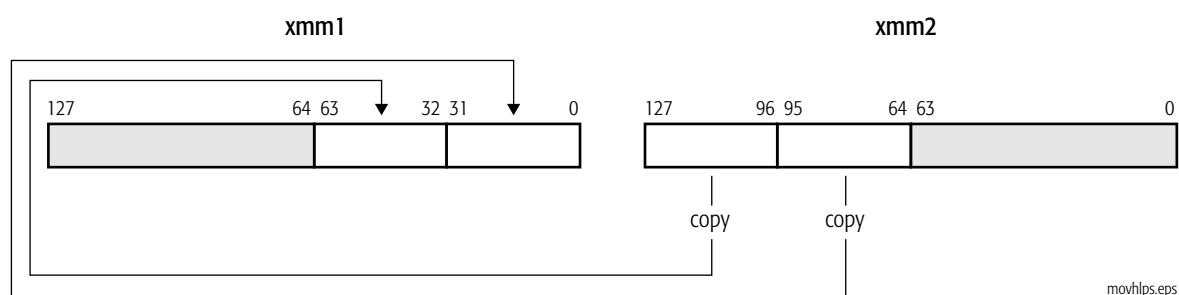
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVHLPS**Move Packed Single-Precision Floating-Point High to Low**

The MOVHLPS instruction moves two packed single-precision floating-point values from the high-order 64 bits of an XMM register to the low-order 64 bits of another XMM register. The high-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVHLPS <i>xmm1</i> , <i>xmm2</i>	0F 12 /r	Moves two packed single-precision floating-point values from an XMM register to the destination XMM register.

**Related Instructions**

MOVAPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVSS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

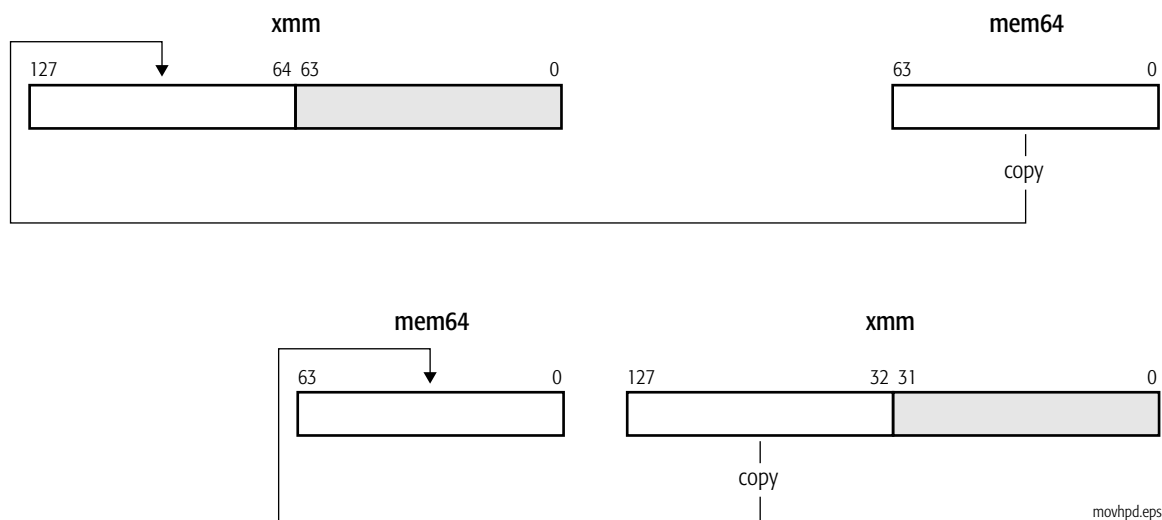
MOVHPD Move High Packed Double-Precision Floating-Point

The MOVHPD instruction moves a double-precision floating-point value:

- from a 64-bit memory location to the high-order 64 bits of an XMM register, or
- from the high-order 64 bits of an XMM register to a 64-bit memory location.

The low-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVHPD <i>xmm</i> , <i>mem64</i>	66 0F 16 /r	Moves double-precision floating-point value from a 64-bit memory location to an XMM register.
MOVHPD <i>mem64</i> , <i>xmm</i>	66 0F 17 /r	Moves double-precision floating-point value from an XMM register to a 64-bit memory location.



Related Instructions

MOVAPD, MOVLPD, MOVMSKPD, MOVSD, MOVUPD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

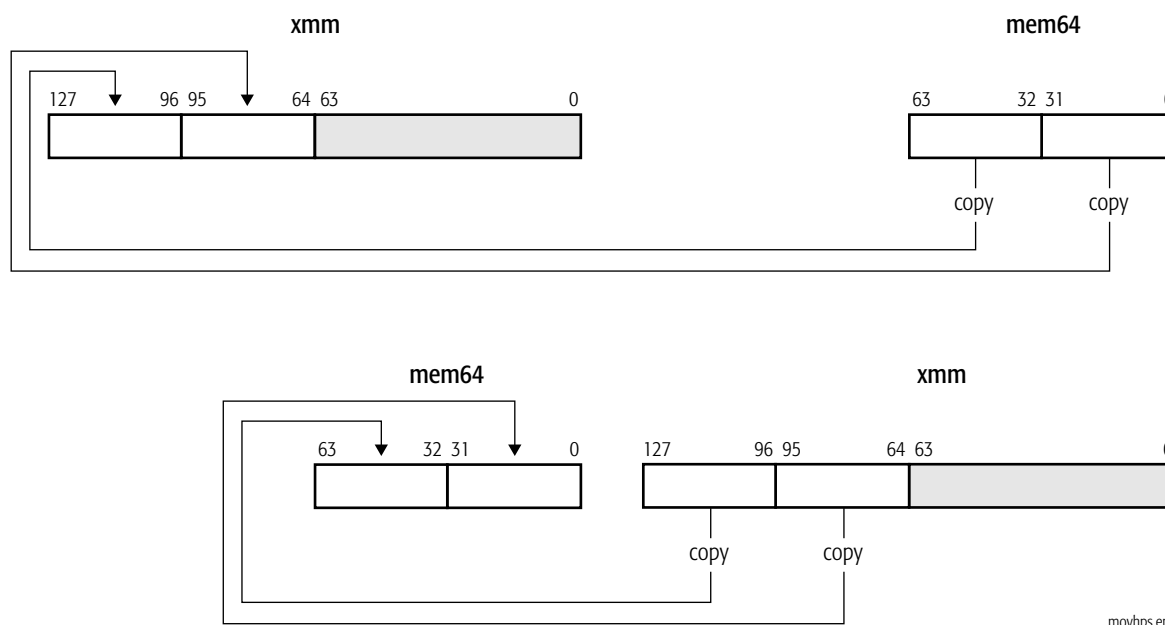
MOVHPS Move High Packed Single-Precision Floating-Point

The MOVHPS instruction moves two packed single-precision floating-point values:

- from a 64-bit memory location to the high-order 64 bits of an XMM register, or
- from the high-order 64 bits of an XMM register to a 64-bit memory location.

The low-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVHPS <i>xmm, mem64</i>	0F 16/ <i>r</i>	Moves two packed single-precision floating-point values from a 64-bit memory location to an XMM register.
MOVHPS <i>mem64, xmm</i>	0F 17/ <i>r</i>	Moves two packed single-precision floating-point values from an XMM register to a 64-bit memory location.



Related Instructions

MOVAPS, MOVHLPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVSS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

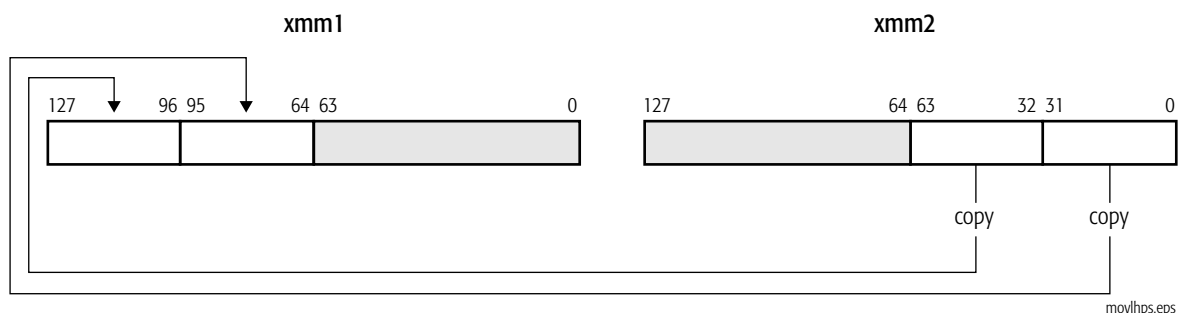
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVLHPS**Move Packed Single-Precision Floating-Point Low to High**

The MOVLHPS instruction moves two packed single-precision floating-point values from the low-order 64 bits of an XMM register to the high-order 64 bits of another XMM register. The low-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVLHPS <i>xmm1, xmm2</i>	0F 16/r	Moves two packed single-precision floating-point values from an XMM register to another XMM register.

**Related Instructions**

MOVAPS, MOVHLPS, MOVHPS, MOVLPS, MOVMSKPS, MOVSS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

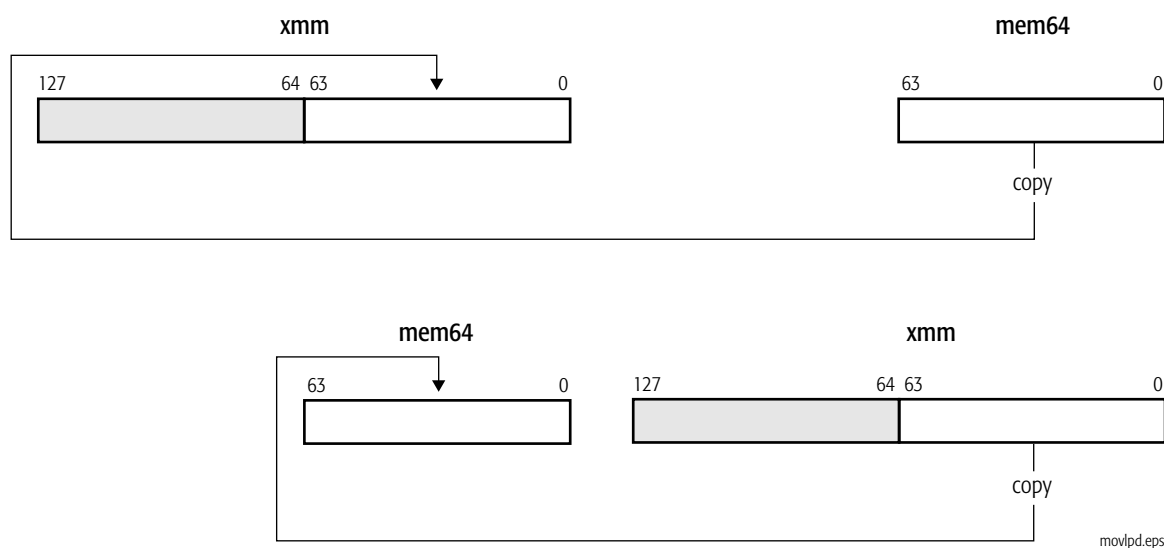
MOVLPD Move Low Packed Double-Precision Floating-Point

The MOVLPD instruction moves a double-precision floating-point value:

- from a 64-bit memory location to the low-order 64 bits of an XMM register, or
- from the low-order 64 bits of an XMM register to a 64-bit memory location.

The high-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVLPD <i>xmm, mem64</i>	66 0F 12 /r	Moves double-precision floating-point value from a 64-bit memory location to an XMM register.
MOVLPD <i>mem64, xmm</i>	66 0F 13 /r	Moves double-precision floating-point value from an XMM register to a 64-bit memory location.



Related Instructions

MOVAPD, MOVHPD, MOVMSKPD, MOVSD, MOVUPD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

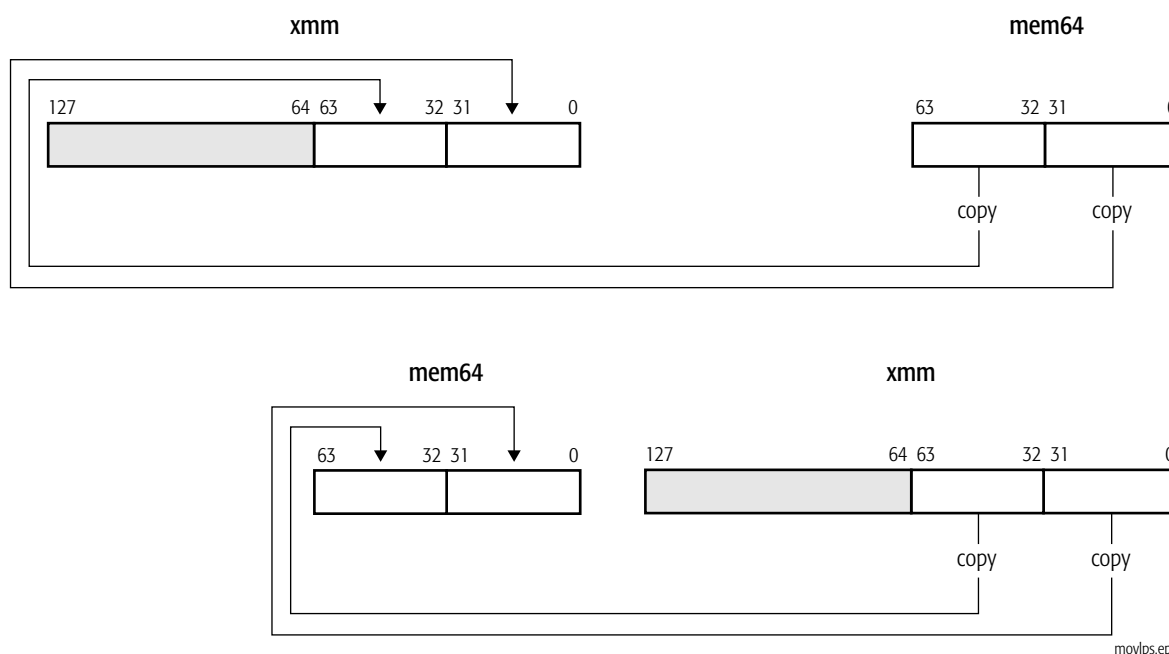
MOVLPS Move Low Packed Single-Precision Floating-Point

The MOVLPS instruction moves two packed single-precision floating-point values:

- from a 64-bit memory location to the low-order 64 bits of an XMM register, or
- from the low-order 64 bits of an XMM register to a 64-bit memory location

The high-order 64 bits of the destination XMM register are not modified.

Mnemonic	Opcode	Description
MOVLPS <i>xmm, mem64</i>	0F 12/r	Moves two packed single-precision floating-point values from a 64-bit memory location to an XMM register.
MOVLPS <i>mem64, xmm</i>	0F 13/r	Moves two packed single-precision floating-point values from an XMM register to a 64-bit memory location.



Related Instructions

MOVAPS, MOVHLPS, MOVHPS, MOVLHPS, MOVMSKPS, MOVSS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

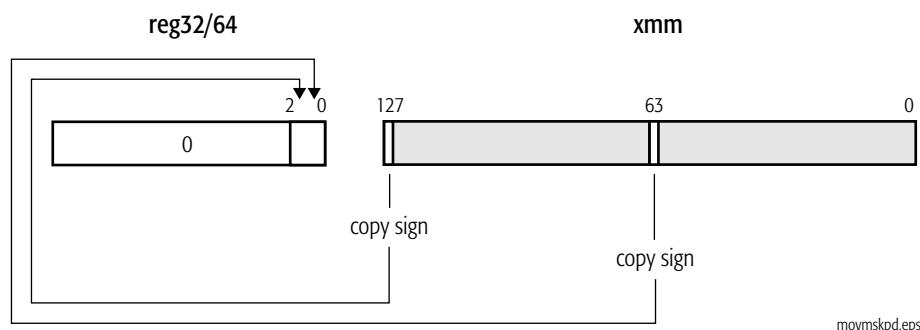
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of the control register (CR4) is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		One of the data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVMSKPD**Extract Packed Double-Precision Floating-Point Sign Mask**

The MOVMSKPD instruction moves the sign bits of two packed double-precision floating-point values in an XMM register to the two low-order bits of a general-purpose register, with zero-extension.

Mnemonic	Opcode	Description
MOVMSKPD <i>reg32, xmm</i>	66 0F 50 /r	Move sign bits in an XMM register to a 32-bit general-purpose register.
MOVMSKPD <i>reg64, xmm</i>	66 0F 50 /r	Move sign bits in an XMM register to a 64-bit general-purpose register.

**Related Instructions**

MOVMSKPS, PMOVMSKB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

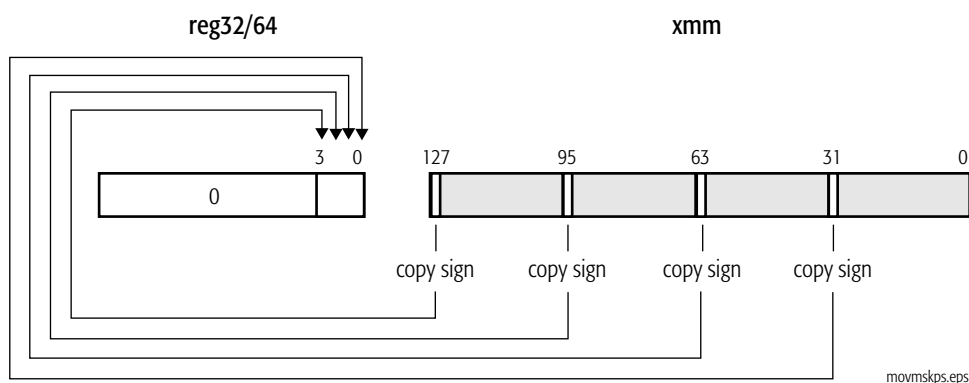
Exceptions

Exception (vector)	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

MOVMSKPS**Extract Packed Single-Precision Floating-Point Sign Mask**

The MOVMSKPD instruction moves the sign bits of four packed single-precision floating-point values in an XMM register to the four low-order bits of a general-purpose register, with zero-extension.

Mnemonic	Opcode	Description
MOVMSKPS <i>reg32, xmm</i>	0F 50 /r	Move sign bits in an XMM register to a 32-bit general-purpose register.
MOVMSKPS <i>reg64, xmm</i>	0F 50 /r	Move sign bits in an XMM register to a 64-bit general-purpose register.

**Related Instructions**

MOVMSKPD, PMOVMSKB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

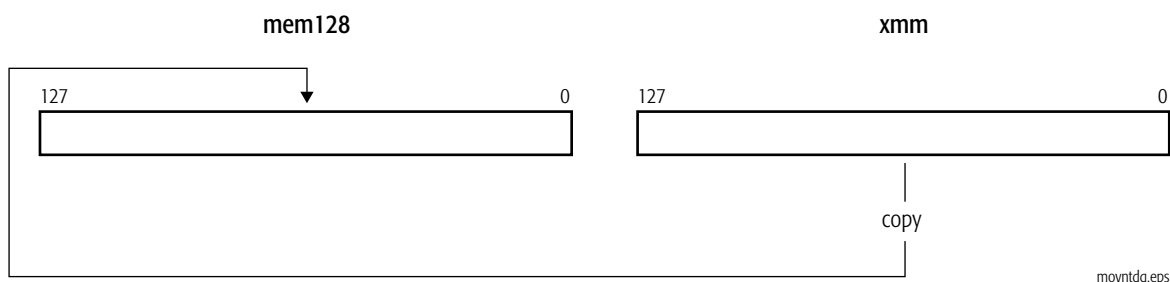
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

MOVNTDQ**Move Non-Temporal Double Quadword**

The MOVNTDQ instruction stores a 128-bit (double quadword) XMM register value into a 128-bit memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see Section 3.9, “Memory Optimization”, on page 115 of Volume 1.

MOVNTDQ is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTDQ with respect to other stores.

Mnemonic	Opcode	Description
MOVNTDQ <i>mem128, xmm</i>	66 0F E7 /r	Stores a 128-bit XMM register value into a 128-bit memory location, minimizing cache pollution.



movntdq.eps

Related Instructions

MOVNTI, MOVNTPD, MOVNTPS, MOVNTQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

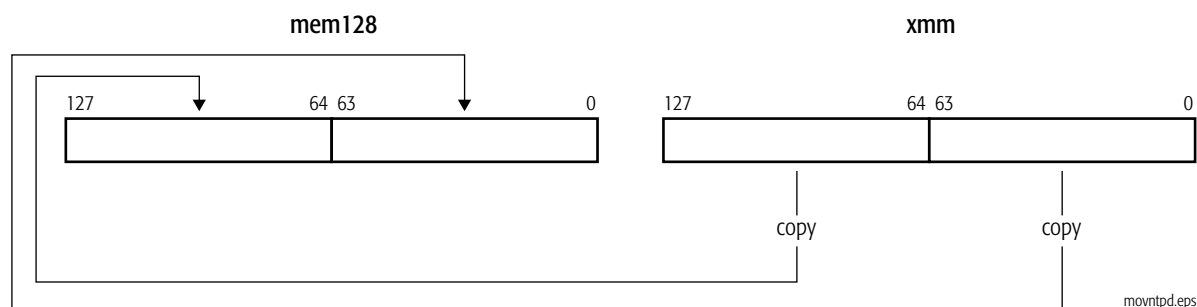
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (CR0.EM) is set to 1. The operating-system FXSAVE/FXRSTOR support bit (CR4.OSFXSR) is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (CR0.TS) is set to 1.
Stack, #SS			X	The operand's effective address points to an illegal memory location in the SS segment.
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, segment overrun, #GP	X	X		The effective address of a data operand is outside the address range 00000h to 0FFFFh.
General protection, #GP			X	The operand's effective address points to an illegal memory location in the CS, DS, ES, FS, or GS segment.
Page fault, #PF		X	X	A page fault resulted from executing the instruction.

MOVNTPD**Move Non-Temporal Packed Double-Precision Floating-Point**

The MOVNTPD instruction stores two double-precision floating-point XMM register values into a 128-bit memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see Section 3.9, “Memory Optimization”, on page 115 of Volume 1.

Mnemonic	Opcode	Description
MOVNTPD <i>mem128, xmm</i>	66 0F 2B /r	Stores two packed double-precision floating-point XMM register values into a 128-bit memory location, minimizing cache pollution.



MOVNTPD is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTPD with respect to other stores.

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPS, MOVNTQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

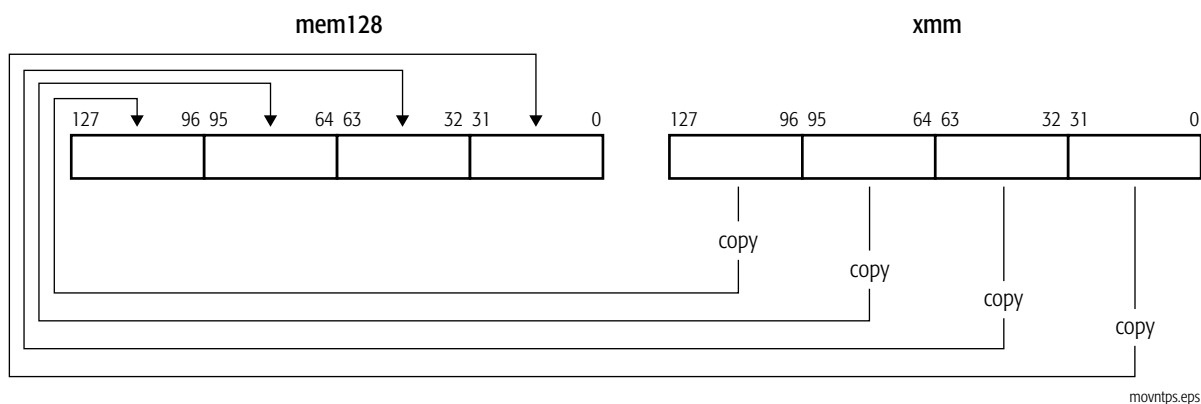
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (CR0.EM) is set to 1. The operating-system FXSAVE/FXRSTOR support bit (CR4.OSFXSR) is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (CR0.TS) is set to 1.
Stack, #SS			X	The operand's effective address points to an illegal memory location in the SS segment.
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, segment overrun, #GP	X	X		The effective address of a data operand is outside the address range 00000h to 0FFFFh.
General protection, #GP			X	The operand's effective address points to an illegal memory location in the CS, DS, ES, FS, or GS segment.
Page fault, #PF		X	X	A page fault resulted from executing the instruction.

MOVNTPS**Move Non-Temporal Packed
Single-Precision Floating-Point**

The MOVNTPS instruction stores four single-precision floating-point XMM register values into a 128-bit memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see Section 3.9, “Memory Optimization”, on page 115 of Volume 1.

Mnemonic	Opcode	Description
MOVNTPS <i>mem128, xmm</i>	0F 2B /r	Stores four packed single-precision floating-point XMM register values into a 128-bit memory location, minimizing cache pollution.



MOVNTPD is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTPD with respect to other stores.

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPD, MOVNTQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

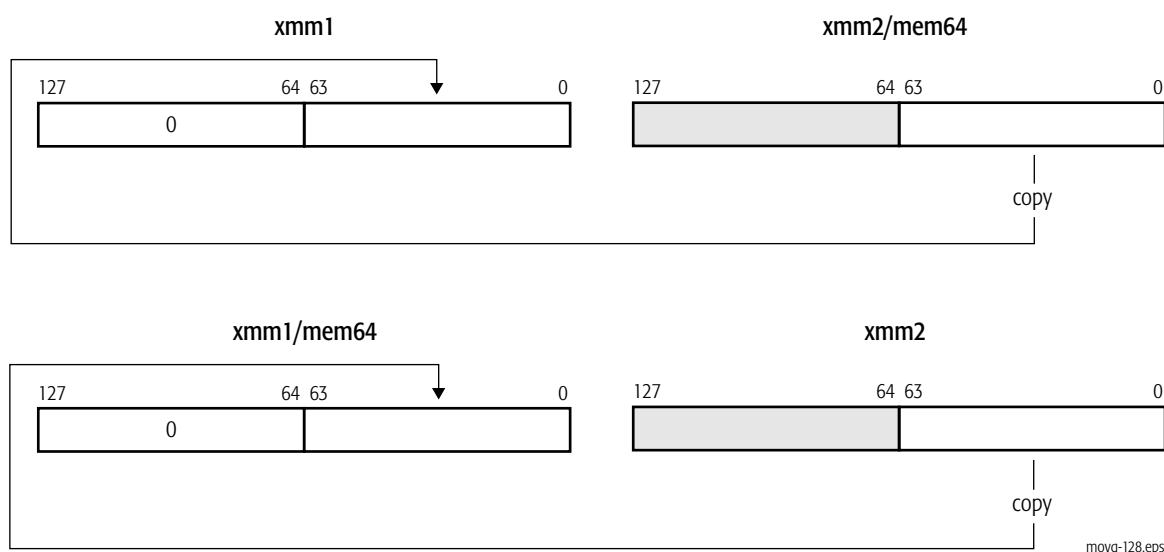
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (CR0.EM) is set to 1. The operating-system FXSAVE/FXRSTOR support bit (CR4.OSFXSR) is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (CR0.TS) is set to 1.
Stack, #SS			X	The operand's effective address points to an illegal memory location in the SS segment.
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, segment overrun, #GP	X	X		The effective address of a data operand is outside the address range 00000h to 0FFFFh.
General protection, #GP			X	The operand's effective address points to an illegal memory location in the CS, DS, ES, FS, or GS segment.
Page fault, #PF		X	X	A page fault resulted from executing the instruction.

MOVQ Move Quadword

The MOVQ instruction moves a 64-bit value:

- from the low-order 64 bits of an XMM register or a 64-bit memory location to the low-order 64 bits of another XMM register, with zero-extension to 128 bits, or
- from the low-order 64 bits of an XMM register to the low-order 64 bits of another XMM register, with zero-extension to 128 bits or to a 64-bit memory location, or

Mnemonic	Opcode	Description
MOVQ <i>xmm1, xmm2/mem64</i>	F3 0F 7E	Moves 64-bit value from an XMM register or memory location to an XMM register.
MOVQ <i>xmm1/mem64, xmm2</i>	66 0F D6	Moves 64-bit value from an XMM register to an XMM register or memory location.



Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

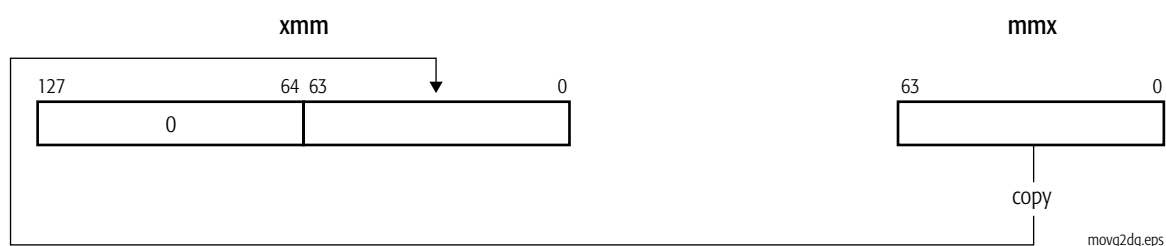
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVQ2DQ**Move Quadword to Quadword**

The MOVQ2DQ instruction moves a 64-bit value from an MMX register to the low-order 64 bits of an XMM register, with zero-extension to 128 bits.

Mnemonic	Opcode	Description
MOVQ2DQ <i>xmm, mmx</i>	F3 0F D6	Moves 64-bit value from an MMX register to an XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

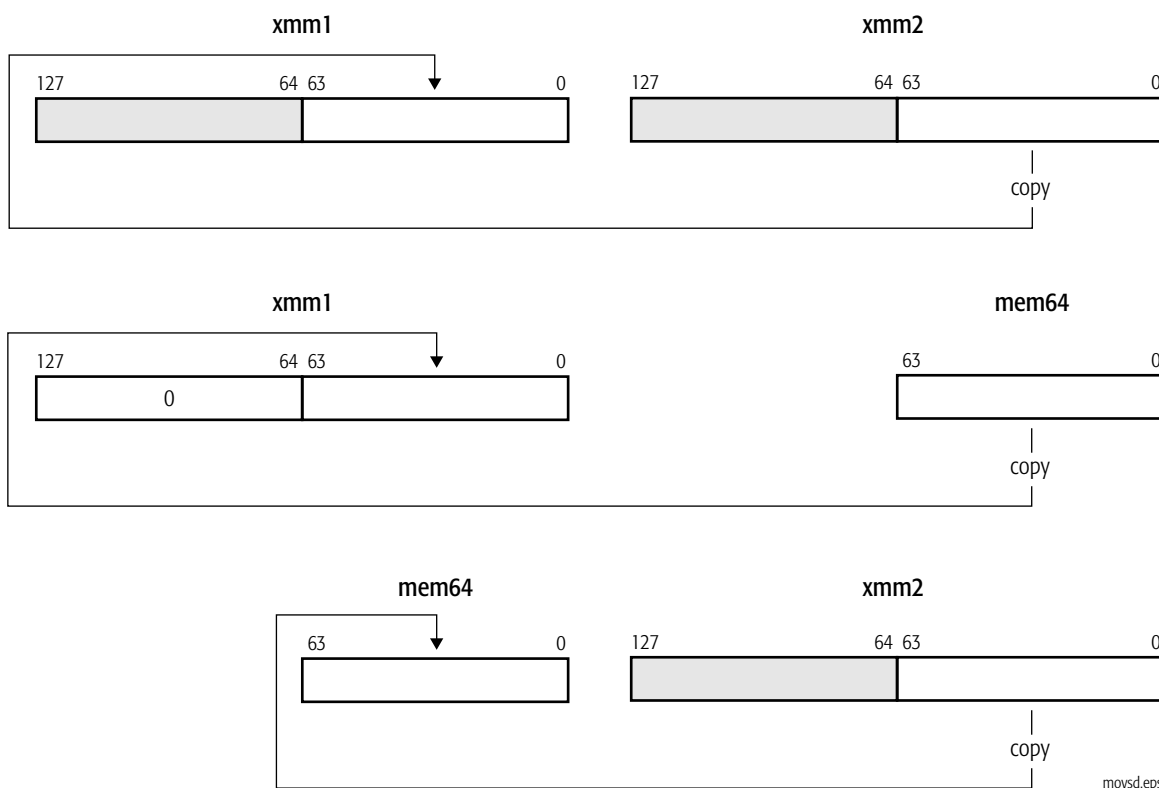
MOVSD Move Scalar Double-Precision Floating-Point

The MOVSD instruction moves a scalar double-precision floating-point value:

- from the low-order 64 bits of an XMM register or a 64-bit memory location to the low-order 64 bits of another XMM register, or
- from the low-order 64 bits of an XMM register to the low-order 64 bits of another XMM register or a 64-bit memory location.

If the source operand is an XMM register, the high-order 64 bits of the destination XMM register are not modified. If the source operand is a memory location, the high-order 64 bits of the destination XMM register are cleared to all 0s.

Mnemonic	Opcode	Description
MOVSD <i>xmm1</i> , <i>xmm2</i> / <i>mem64</i>	F2 0F 10 /r	Moves double-precision floating-point value from an XMM register or 64-bit memory location to an XMM register.
MOVSD <i>xmm1</i> / <i>mem64</i> , <i>xmm2</i>	F2 0F 11 /r	Moves double-precision floating-point value from an XMM register to an XMM register or 64-bit memory location.



This MOVSD instruction should not be confused with the same-mnemonic MOVSD (move string doubleword) instruction in the general-purpose instruction set. Assemblers can distinguish the instructions by the number and type of operands.

Related Instructions

MOVAPD, MOVHPD, MOVLPD, MOVMSKPD, MOVUPD

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

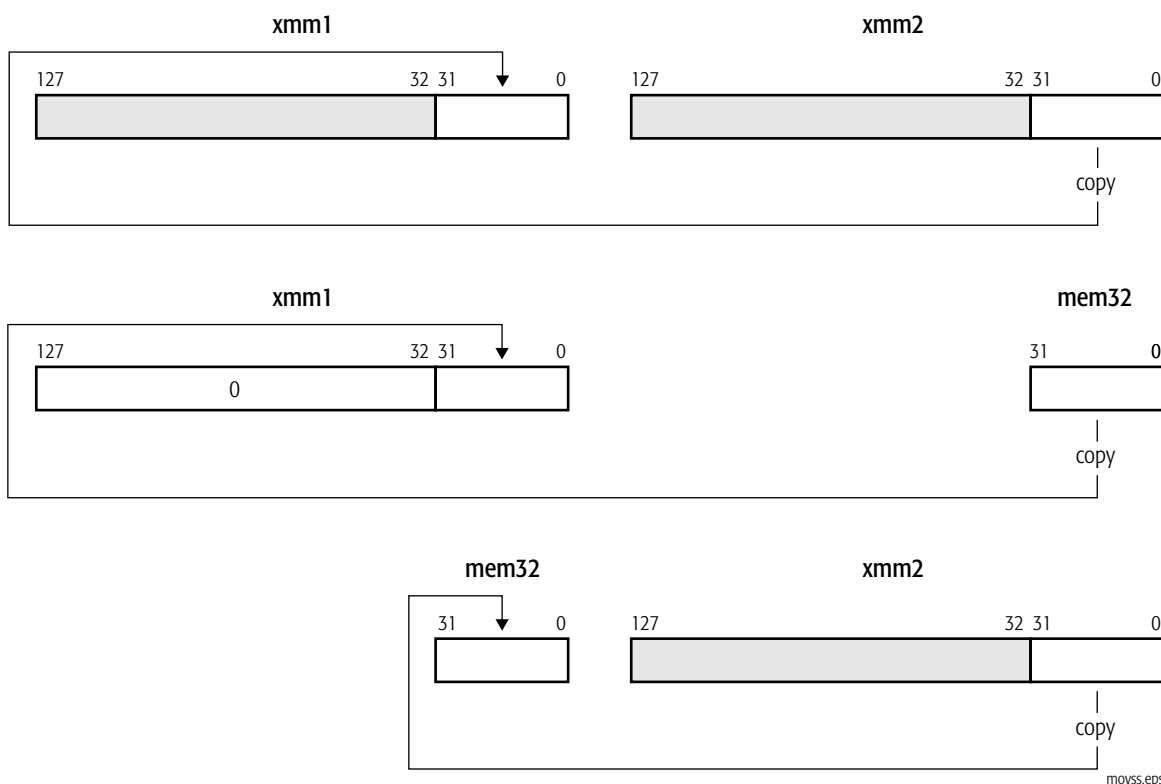
MOVSS Move Scalar Single-Precision Floating-Point

The MOVSS instruction moves a scalar single-precision floating-point value:

- from the low-order 32 bits of an XMM register or a 32-bit memory location to the low-order 32 bits of another XMM register, or
- from a 32-bit memory location to the low-order 32 bits of an XMM register, with zero-extension to 128 bits.

If the source operand is an XMM register, the high-order 96 bits of the destination XMM register are not modified. If the source operand is a memory location, the high-order 96 bits of the destination XMM register are cleared to all 0s.

Mnemonic	Opcode	Description
MOVSS <i>xmm1, xmm2/mem32</i>	F3 0F 10 /r	Moves single-precision floating-point value from an XMM register or 32-bit memory location to an XMM register.
MOVSS <i>xmm1/mem32, xmm2</i>	F3 0F 11 /r	Moves single-precision floating-point value from an XMM register to an XMM register or 32-bit memory location.



Related Instructions

MOVAPS, MOVHLPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVUPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

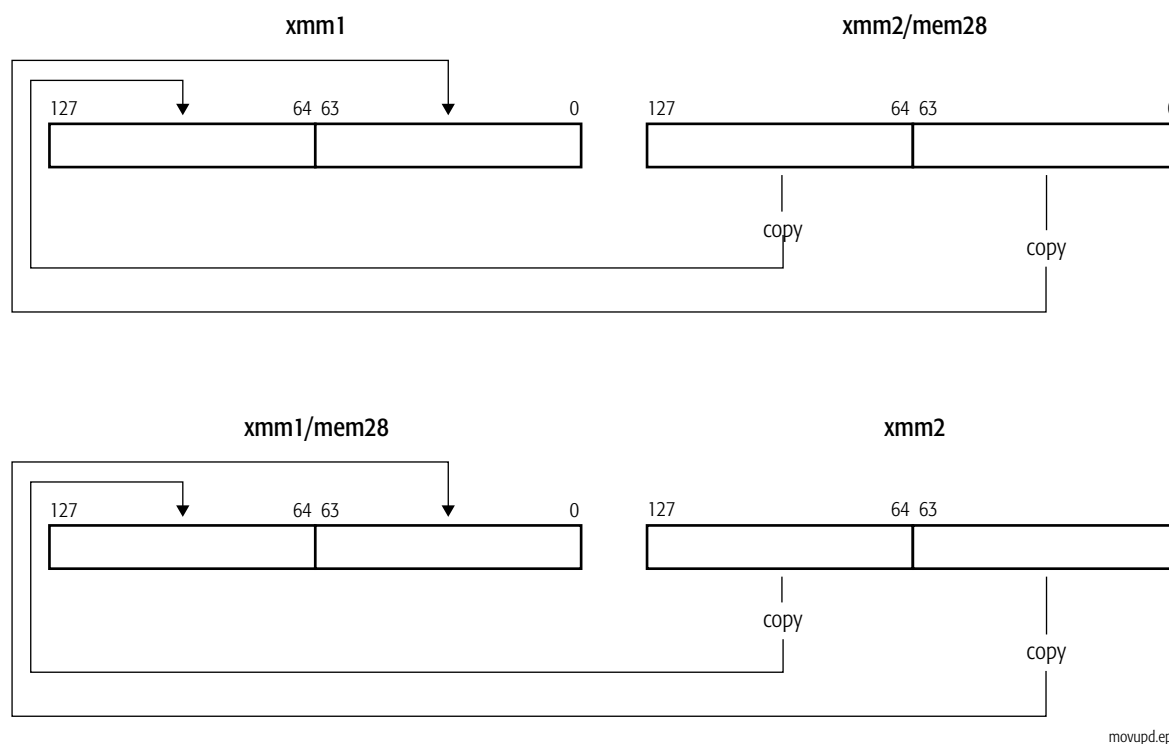
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVUPD Move Unaligned Packed Double-Precision Floating-Point

The MOVUPD instruction moves two packed double-precision floating-point values:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MOVUPD <i>xmm1</i> , <i>xmm2/mem128</i>	66 0F 10 /r	Moves two unaligned packed double-precision floating-point values from an XMM register or 128-bit memory location to an XMM register.
MOVUPD <i>xmm1/mem128</i> , <i>xmm2</i>	66 0F 11 /r	Moves two unaligned packed double-precision floating-point values from an XMM register to an XMM register or 128-bit memory location.



Memory operands that are not aligned on a 16-byte boundary do not cause a general-protection exception.

Related Instructions

MOVAPD, MOVHPD, MOVLPD, MOVMSKPD, MOVSD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

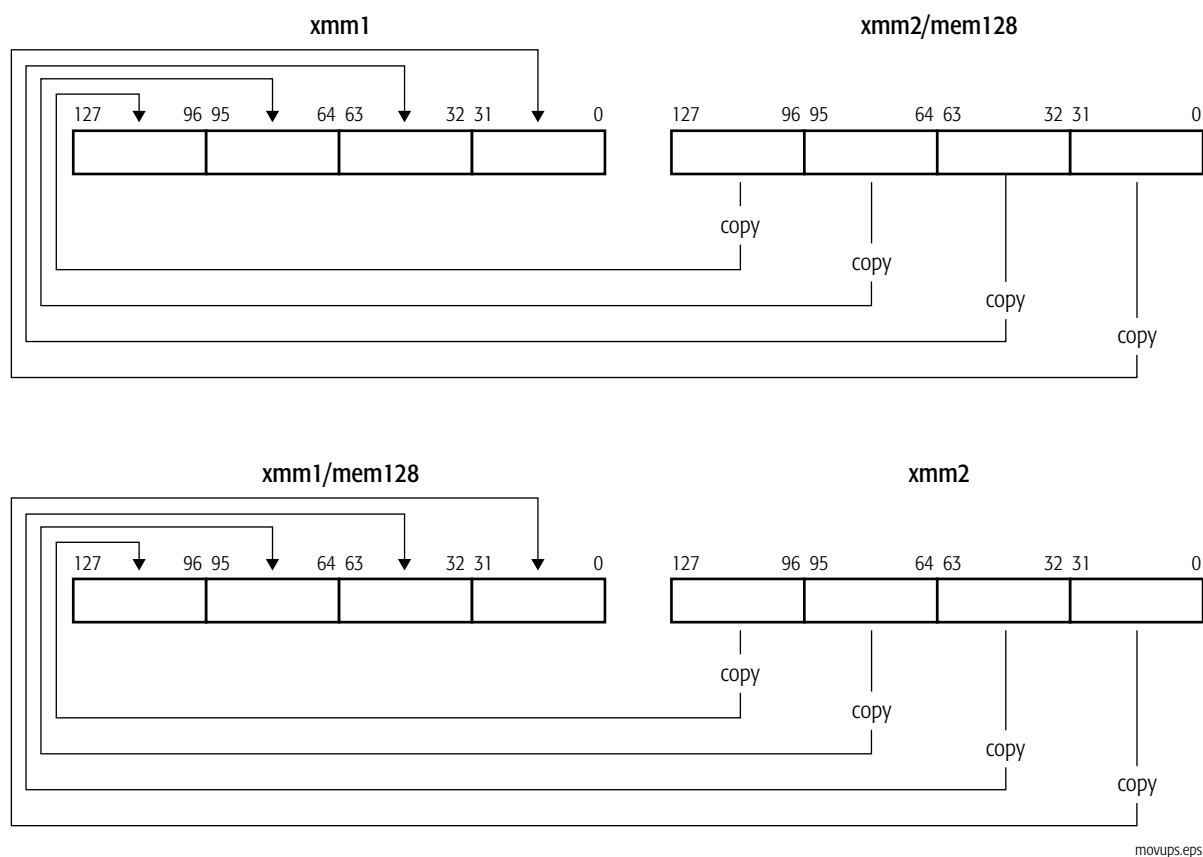
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MOVUPS**Move Unaligned Packed Single-Precision Floating-Point**

The MOVUPS instruction moves four packed single-precision floating-point values:

- from an XMM register or 128-bit memory location to another XMM register, or
- from an XMM register to another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MOVUPS <i>xmm1</i> , <i>xmm2/mem128</i>	0F 10 /r	Moves four unaligned packed single-precision floating-point values from an XMM register or 128-bit memory location to an XMM register.
MOVUPS <i>xmm1/mem128</i> , <i>xmm2</i>	0F 11 /r	Moves four unaligned packed single-precision floating-point values from an XMM register to an XMM register or 128-bit memory location.



Memory operands that are not aligned on a 16-byte boundary do not cause a general-protection exception.

Related Instructions

MOVAPS, MOVHLPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

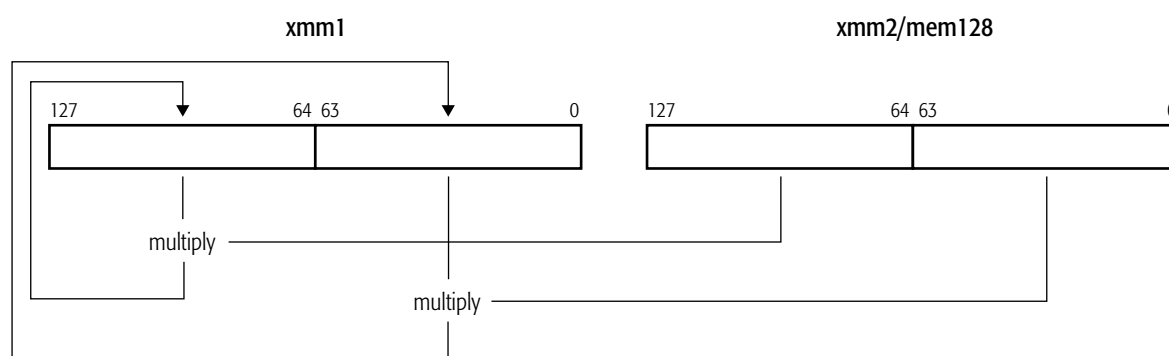
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

MULPD**Multiply Packed Double-Precision Floating-Point**

The MULPD instruction multiplies each of the two packed double-precision floating-point values in the first source operand by the corresponding packed double-precision floating-point value in the second source operand and writes the result of each multiply in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MULPD <i>xmm1, xmm2/mem128</i>	66 0F 59 /r	Multiplies packed double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the results in the destination XMM register.



mulpd.eps

Related Instructions

MULPS, MULSD, MULSS, PFMUL

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.

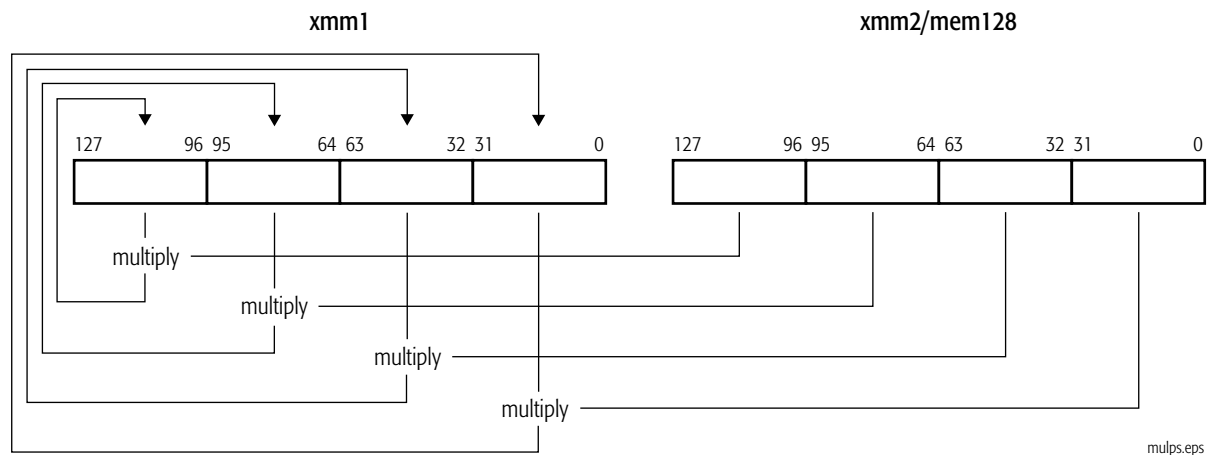
Exception	Real	Virtual 8086	Protected	Cause of Exception
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

MULPS

Multiply Packed Single-Precision Floating-Point

The MULPS instruction multiplies each of the four packed single-precision floating-point values in first source operand by the corresponding packed single-precision floating-point value in the second source operand and writes the result of each multiply in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
MULPS <i>xmm1, xmm2/mem128</i>	0F 59 /r	Multiplies packed single-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and writes the results in the destination XMM register.



Related Instructions

MULPD, MULSD, MULSS, PFMUL

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

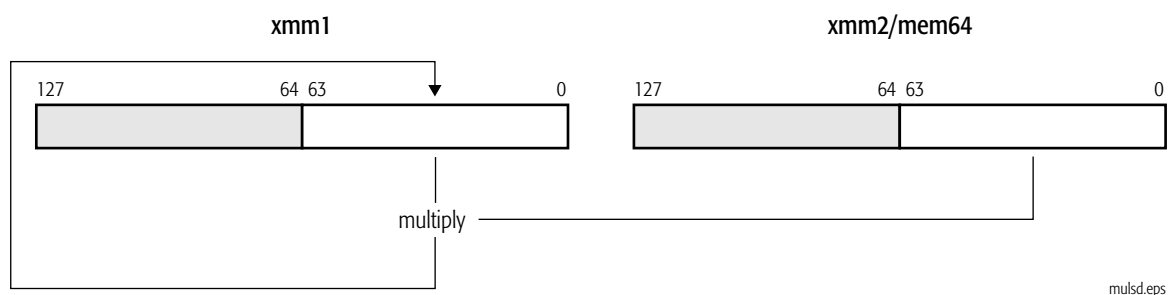
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

MULSD**Multiply Scalar Double-Precision Floating-Point**

The MULSD instruction multiplies the double-precision floating-point value in the low-order quadword of first source operand by the double-precision floating-point value in the low-order quadword of the second source operand and writes the result in the low-order quadword of the destination (first source). The high-order quadword of the destination is not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 64-bit memory location.

Mnemonic	Opcode	Description
MULSD <i>xmm1, xmm2/mem64</i>	F2 0F 59 /r	Multiplies low-order double-precision floating-point values in an XMM register and another XMM register or 64-bit memory location and writes the result in the low-order quadword of the destination XMM register.

**Related Instructions**

MULPD, MULPS, MULSS, PFMUL

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

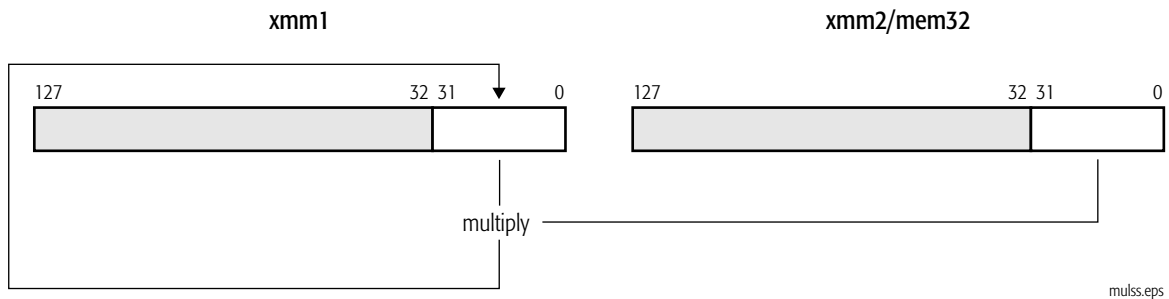
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

MULSS Multiply Scalar Single-Precision Floating-Point

The MULSS instruction multiplies the single-precision floating-point value in the low-order doubleword of first source operand by the single-precision floating-point value in the low-order doubleword of the second source operand and writes the result in the low-order doubleword of the destination (first source). The three high-order doublewords of the destination are not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 32-bit memory location.

Mnemonic	Opcode	Description
MULSS <i>xmm1, xmm2/mem32</i>	F3 0F 59/r	Multiplies low-order single-precision floating-point values in an XMM register and another XMM register or 32-bit memory location and writes the result in the low-order doubleword of the destination XMM register.



mulss.eps

Related Instructions

MULPD, MULPS, MULSD, PFMUL

rFLAGS Affected

None.

MXCSR Flags Affected.

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

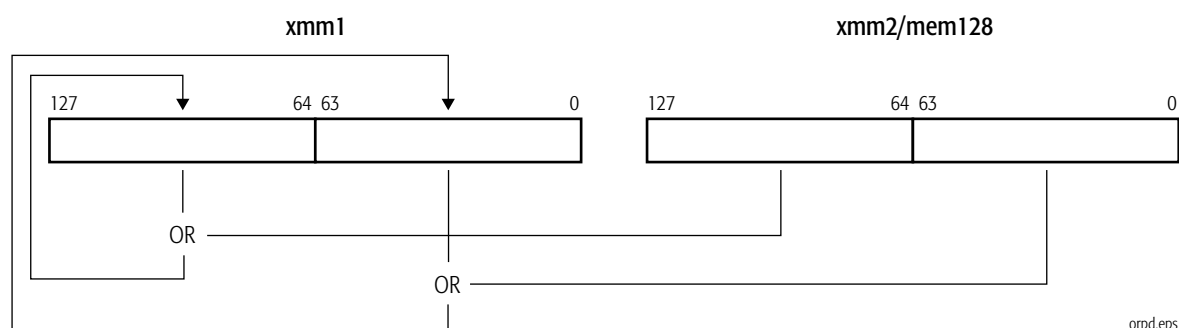
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

ORPD**Logical Bitwise OR
Packed Double-Precision Floating-Point**

The ORPD instruction performs a bitwise logical OR of the two packed double-precision floating-point values in the first source operand and the corresponding two packed double-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ORPD <i>xmm1, xmm2/mem128</i>	66 0F 56 /r	Performs bitwise logical OR of two packed double-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ANDNPD, ANDNPS, ANDPD, ANDPS, ORPS, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

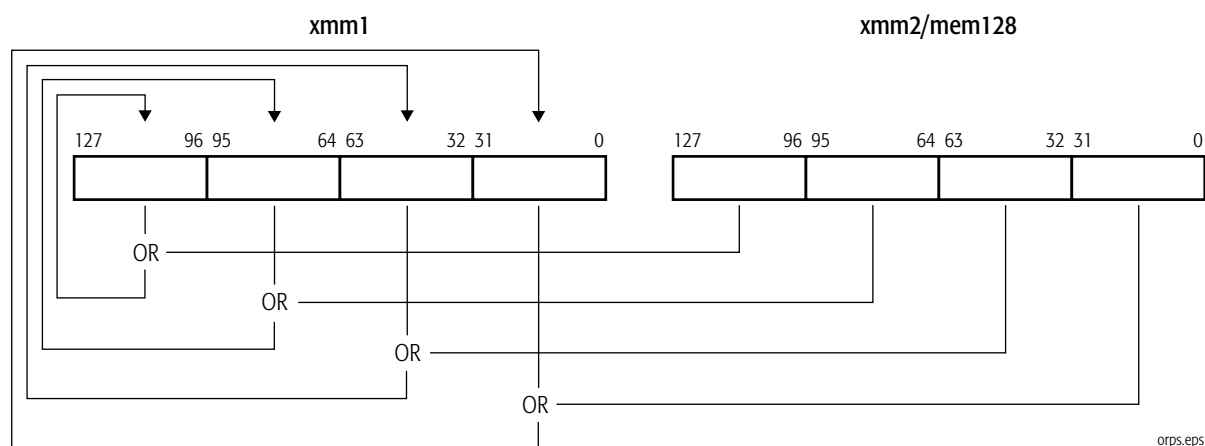
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

ORPS**Logical Bitwise OR
Packed Single-Precision Floating-Point**

The ORPS instruction performs a bitwise logical OR of the four packed single-precision floating-point values in the first source operand and the corresponding four packed single-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
ORPS <i>xmm1, xmm2/mem128</i>	OF 56/r	Performs bitwise logical OR of four packed single-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

ANDNPD, ANDNPS, ANDPD, ANDPS, ORPD, XORPD, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

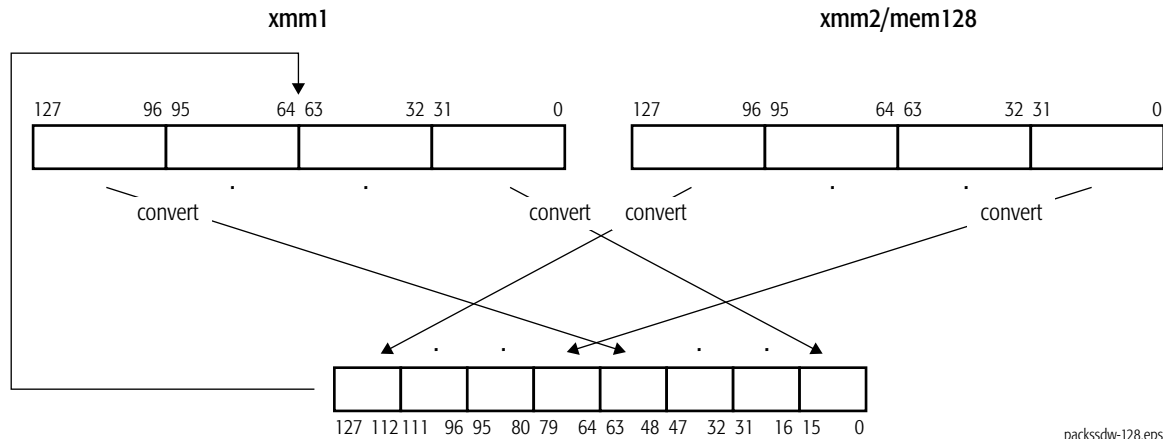
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PACKSSDW**Pack with Saturation Signed Doubleword to Word**

The PACKSSDW instruction converts each 32-bit signed integer in the first and second source operands to a 16-bit signed integer and packs the converted values into words in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Converted values from the first source operand are packed into the low-order words of the destination, and the converted values from the second source operand are packed into the high-order words of the destination.

Mnemonic	Opcode	Description
PACKSSDW <i>xmm1, xmm2/mem128</i>	66 0F 6B /r	Packs 32-bit signed integers in an XMM register and another XMM register or 128-bit memory location into 16-bit signed integers in an XMM register.



packssdw-128.eps

For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PACKSSWB, PACKUSWB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

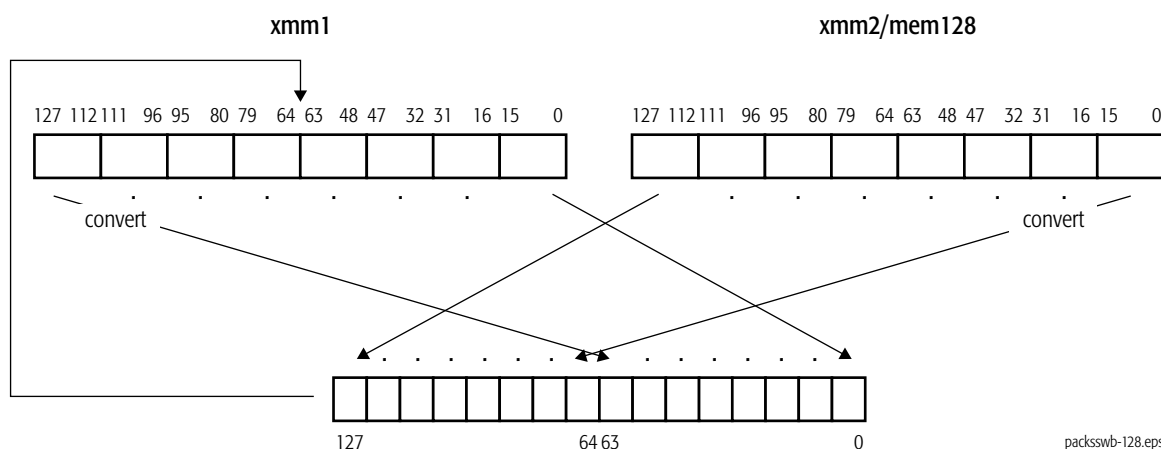
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PACKSSWB**Pack with Saturation Signed Word to Byte**

The PACKSSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit signed integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKSSWB <i>xmm1, xmm2/mem128</i>	66 0F 63 /r	Packs 16-bit signed integers in an XMM register and another XMM register or 128-bit memory location into 8-bit signed integers in an XMM register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PACKSSDW, PACKUSWB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

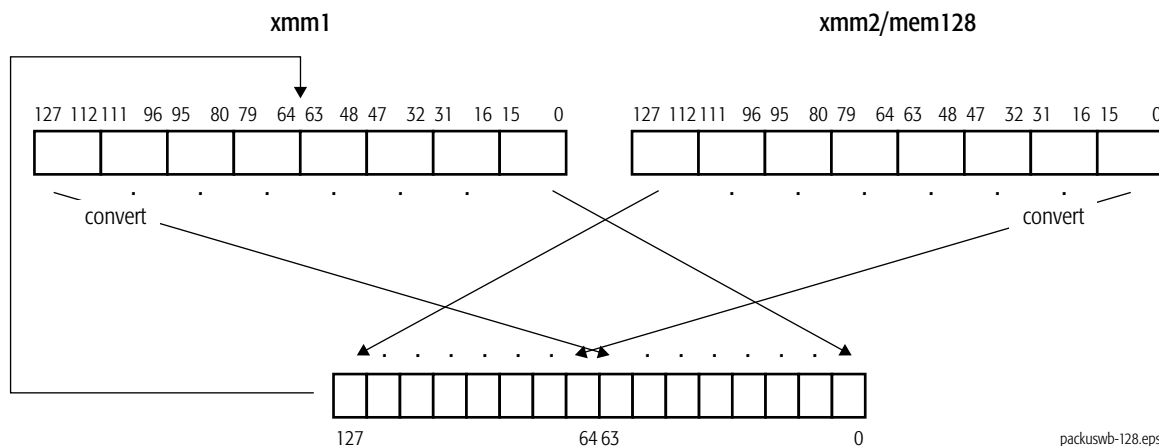
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PACKUSWB**Pack with Saturation Signed Word to Unsigned Byte**

The PACKUSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit unsigned integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKUSWB <i>xmm1, xmm2/mem128</i>	66 0F 67 /r	Packs 16-bit signed integers in an XMM register and another XMM register or 128-bit memory location into 8-bit unsigned integers in an XMM register.



packuswb-128.eps

For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PACKSSDW, PACKSSWB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

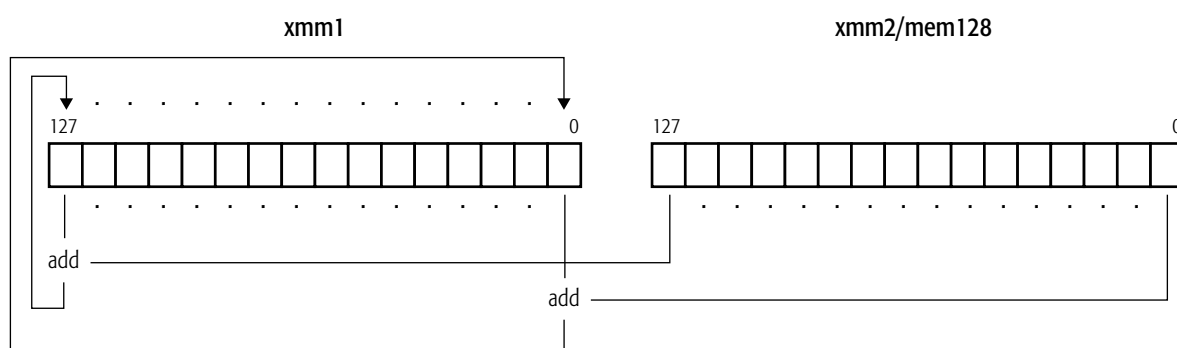
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDB Packed Add Bytes

The PADDB instruction adds each packed 8-bit integer value in the first source operand to the corresponding packed 8-bit integer in the second source operand and writes the integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDB <i>xmm1, xmm2/mem128</i>	66 0F FC /r	Adds packed byte integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



paddb-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

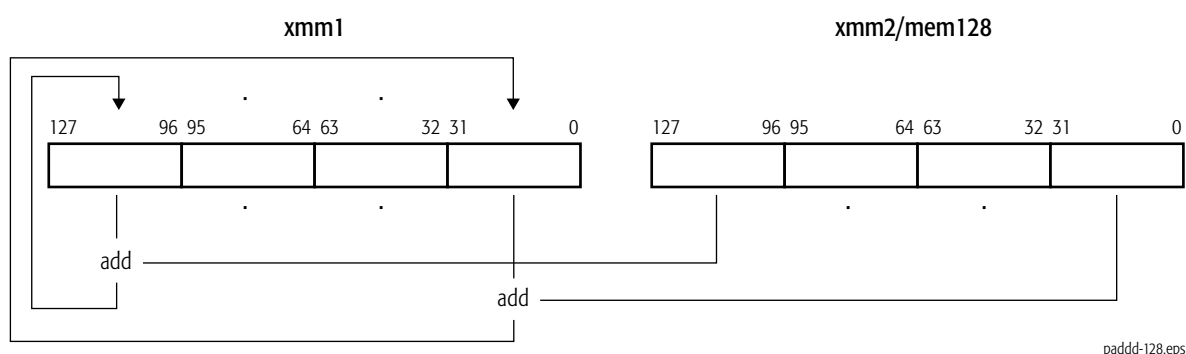
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDD**Packed Add Doublewords**

The PADDD instruction adds each packed 32-bit integer value in the first source operand to the corresponding packed 32-bit integer in the second source operand and writes the integer result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDD <i>xmm1, xmm2/mem128</i>	66 0F FE /r	Adds packed 32-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PADDB, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

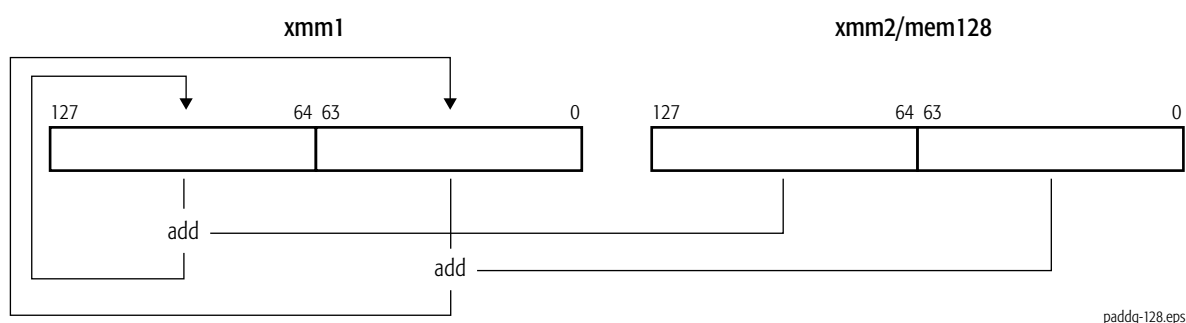
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDQ**Packed Add Quadwords**

The PADDQ instruction adds each packed 64-bit integer value in the first source operand to the corresponding packed 64-bit integer in the second source operand and writes the integer result of each addition in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDQ <i>xmm1, xmm2/mem128</i>	66 0F D4 /r	Adds packed 64-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PADDB, PADDD, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

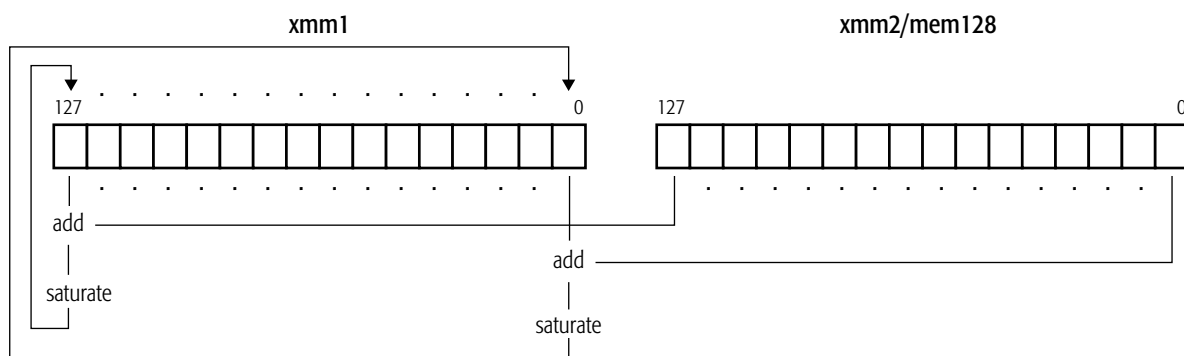
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDSB Packed Add Signed with Saturation Bytes

The PADDSB instruction adds each packed 8-bit signed integer value in the first source operand to the corresponding packed 8-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDSB <i>xmm1, xmm2/mem128</i>	66 0F EC /r	Adds packed byte signed integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



paddsb-128.eps

For each packed value in the destination, if the value is larger than the largest representable signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

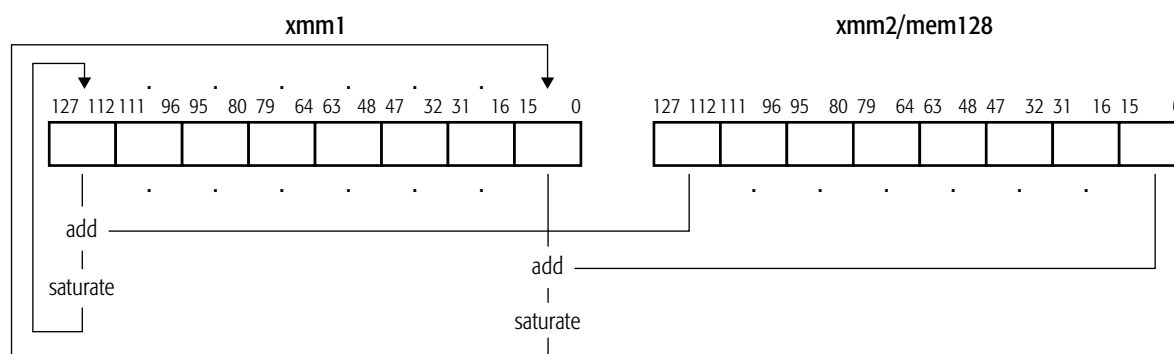
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDSW**Packed Add Signed with Saturation Words**

The PADDSW instruction adds each packed 16-bit signed integer value in the first source operand to the corresponding packed 16-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDSW <i>xmm1, xmm2/mem128</i>	66 0F ED /r	Adds packed 16-bit signed integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



paddsw-128.eps

For each packed value in the destination, if the value is larger than the largest representable signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

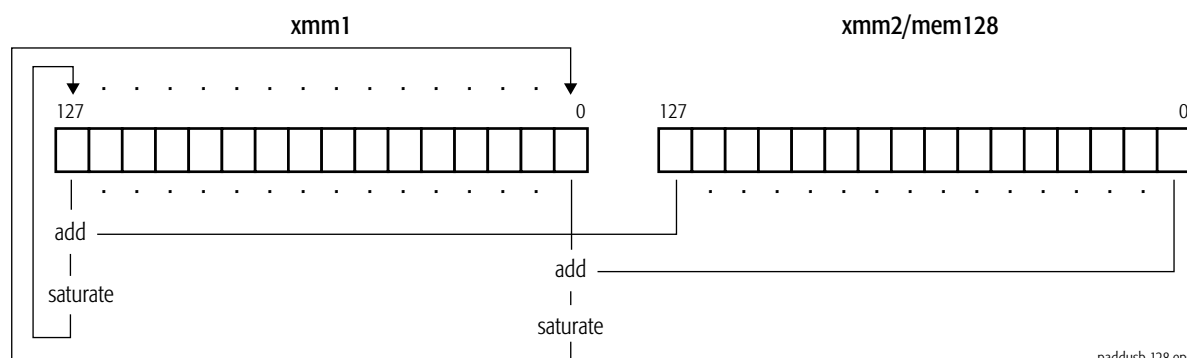
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDUSB**Packed Add Unsigned with Saturation Bytes**

The PADDUSB instruction adds each packed 8-bit unsigned integer value in the first source operand to the corresponding packed 8-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDUSB <i>xmm1, xmm2/mem128</i>	66 0F DC /r	Adds packed byte unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSW, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

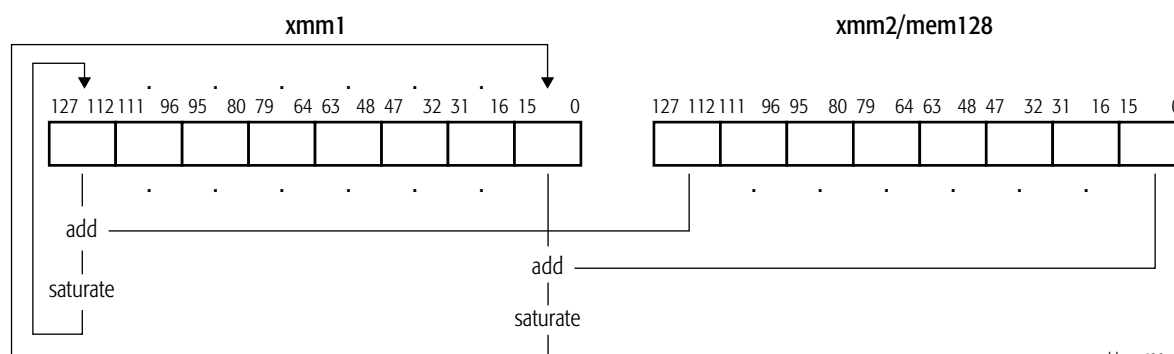
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDUSW**Packed Add Unsigned with Saturation Words**

The PADDUSW instruction adds each packed 16-bit unsigned integer value in the first source operand to the corresponding packed 16-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDUSW <i>xmm1, xmm2/mem128</i>	66 0F DD /r	Adds packed 16-bit unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes result in the destination XMM register.



paddusw-128.eps

For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

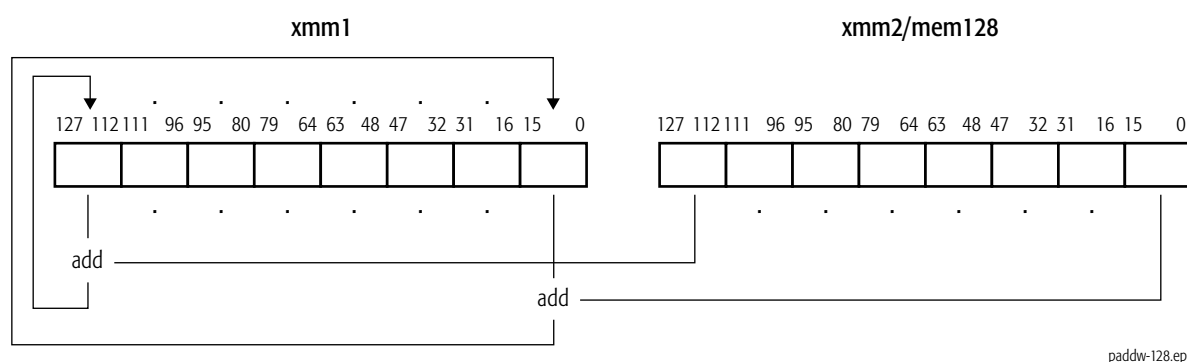
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PADDW Packed Add Words

The PADDW instruction adds each packed 16-bit integer value in the first source operand to the corresponding packed 16-bit integer in the second source operand and writes the integer result of each addition in the corresponding word of the destination (second source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PADDW <i>xmm1, xmm2/mem128</i>	66 0F FD /r	Adds packed 16-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



paddw-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PADB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

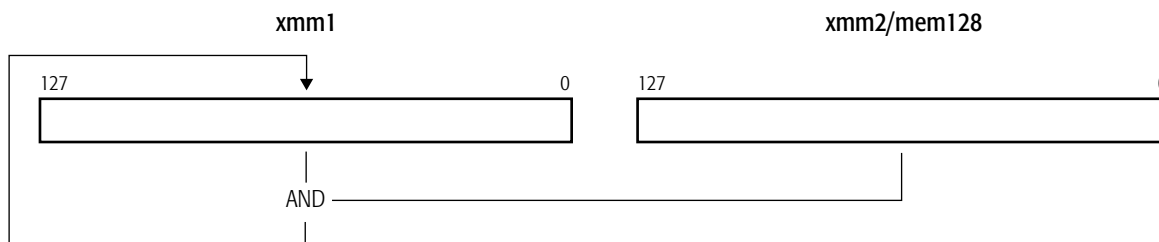
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PAND**Packed Logical Bitwise AND**

The PAND instruction performs a bitwise logical AND of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PAND <i>xmm1, xmm2/mem128</i>	66 0F DB /r	Performs bitwise logical AND of values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



pand-128.eps

Related Instructions

PANDN, POR, PXOR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

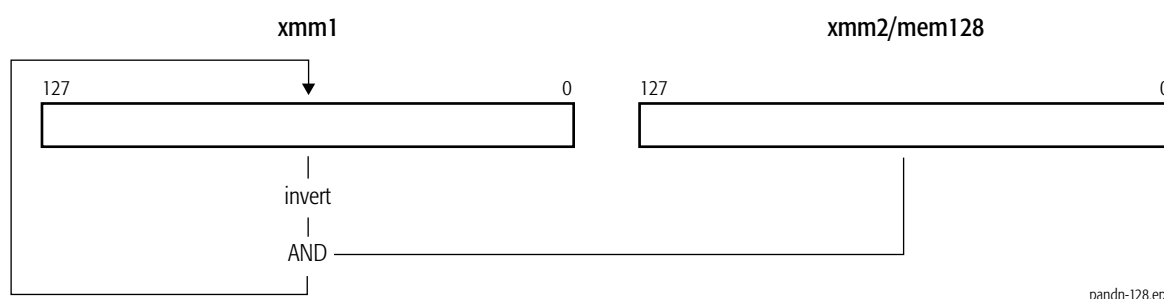
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PANDN**Packed Logical Bitwise AND NOT**

The PANDN instruction performs a bitwise logical AND of the value in the second source operand and the one's complement of the value in the first source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PANDN <i>xmm1, xmm2/mem128</i>	66 0F DF /r	Performs bitwise logical AND NOT of values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

PAND, POR, PXOR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

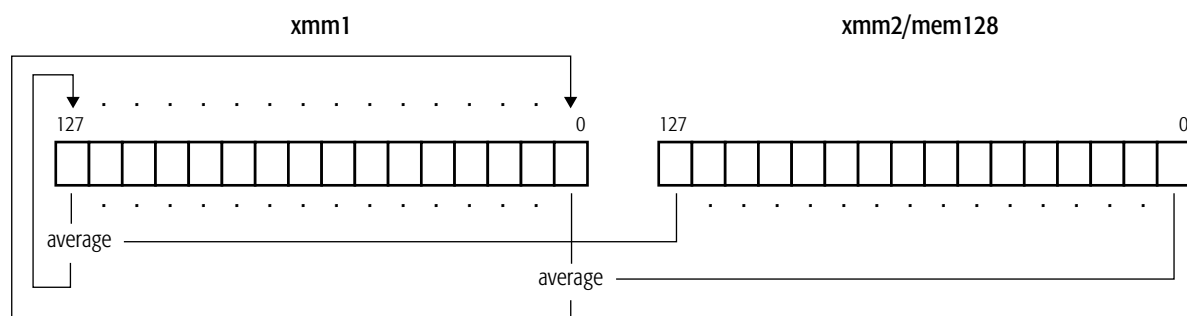
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PAVGB**Packed Average Unsigned Bytes**

The PAVGB instruction computes the rounded average of each packed unsigned 8-bit integer value in the first source operand and the corresponding packed 8-bit unsigned integer in the second source operand and writes each average in the corresponding byte of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 9-bit temporary sum, and then right-shifting the temporary sum by one bit position. The destination and source operands are an XMM register and another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PAVGB <i>xmm1, xmm2/mem128</i>	66 0F E0 /r	Averages packed 8-bit unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



pavgb-128.eps

Related Instructions

PAVGW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

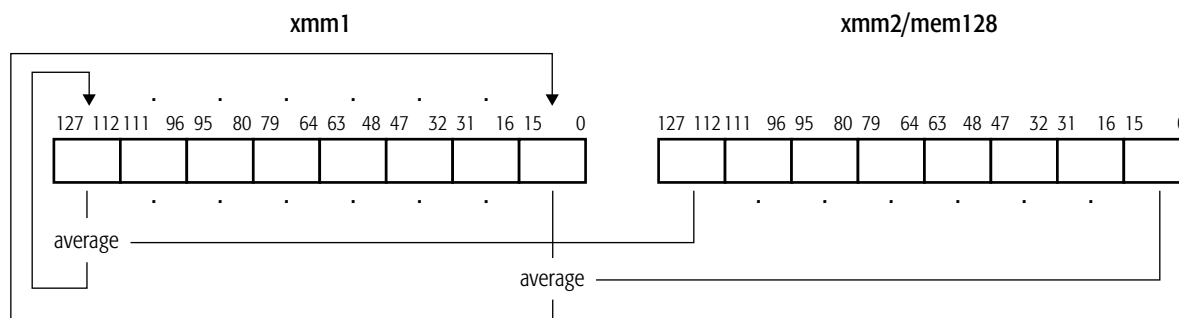
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PAVGW**Packed Average Unsigned Words**

The PAVGW instruction computes the rounded average of each packed unsigned 16-bit integer value in the first source operand and the corresponding packed 16-bit unsigned integer in the second source operand and writes each average in the corresponding word of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 17-bit temporary sum, and then right-shifting the temporary sum by one bit position. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PAVGW <i>xmm1, xmm2/mem128</i>	66 0F E3 /r	Averages packed 16-bit unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes the result in the destination XMM register.



pavgw-128.eps

Related Instructions

PAVGB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

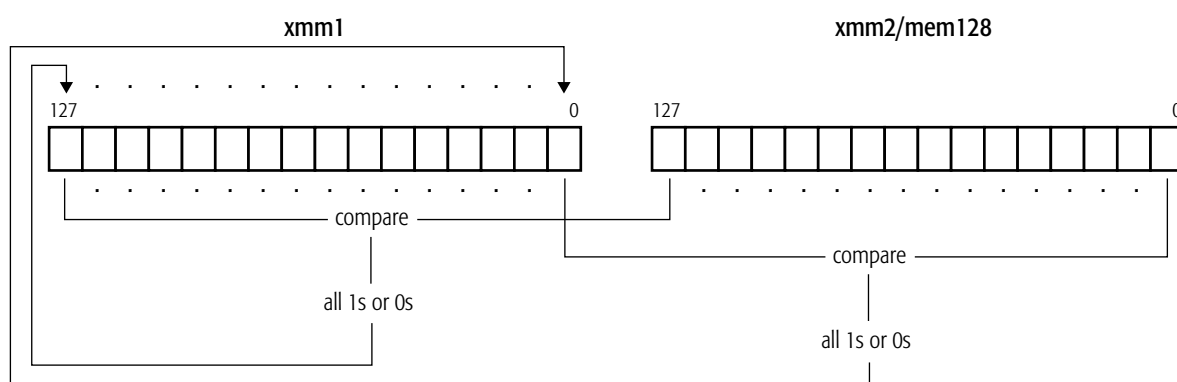
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPEQB**Packed Compare Equal Bytes**

The PCMPEQB instruction compares corresponding packed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPEQB <i>xmm1, xmm2/mem128</i>	66 0F 74 /r	Compares packed bytes in an XMM register and an XMM register or 128-bit memory location.



pcmpeqb-128.eps

Related Instructions

PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

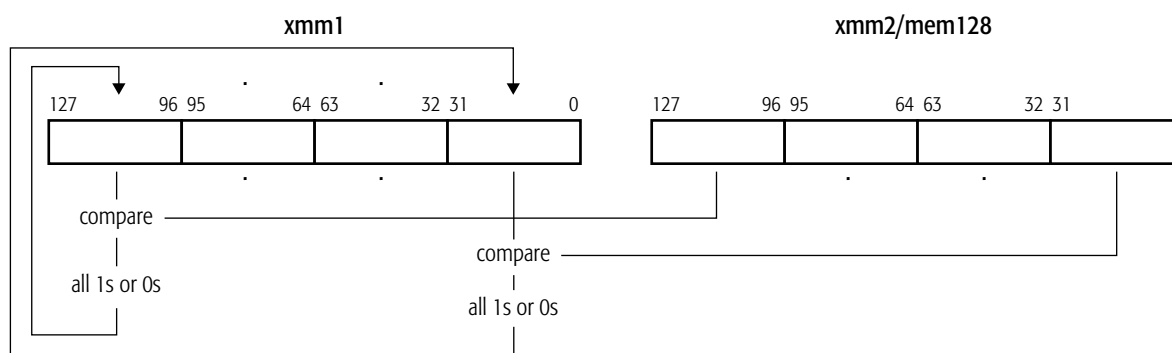
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPEQD**Packed Compare Equal Doublewords**

The PCMPEQD instruction compares corresponding packed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPEQD <i>xmm1, xmm2/mem128</i>	66 0F 76 /r	Compares packed doublewords in an XMM register and an XMM register or 128-bit memory location.



pcmpeq-d-128.eps

Related Instructions

PCMPEQB, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

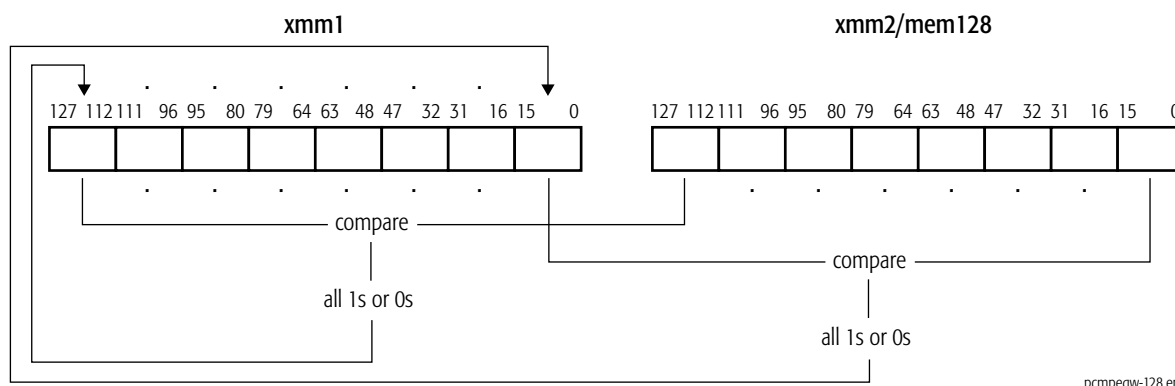
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPEQW**Packed Compare Equal Words**

The PCMPEQW instruction compares corresponding packed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPEQW <i>xmm1, xmm2/mem128</i>	66 0F 75 /r	Compares packed 16-bit values in an XMM register and an XMM register or 128-bit memory location.



pcmpeqw-128.eps

Related Instructions

PCMPEQB, PCMPEQD, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

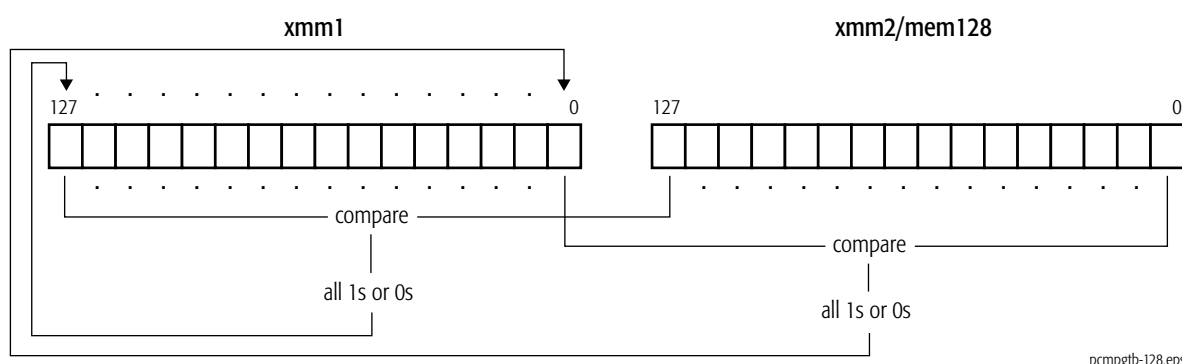
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPGTB**Packed Compare Greater Than Signed Bytes**

The PCMPGTB instruction compares corresponding packed signed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPGTB <i>xmm1, xmm2/mem128</i>	66 0F 64 /r	Compares packed signed bytes in an XMM register and an XMM register or 128-bit memory location.



pcmpgtb-128.eps

Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

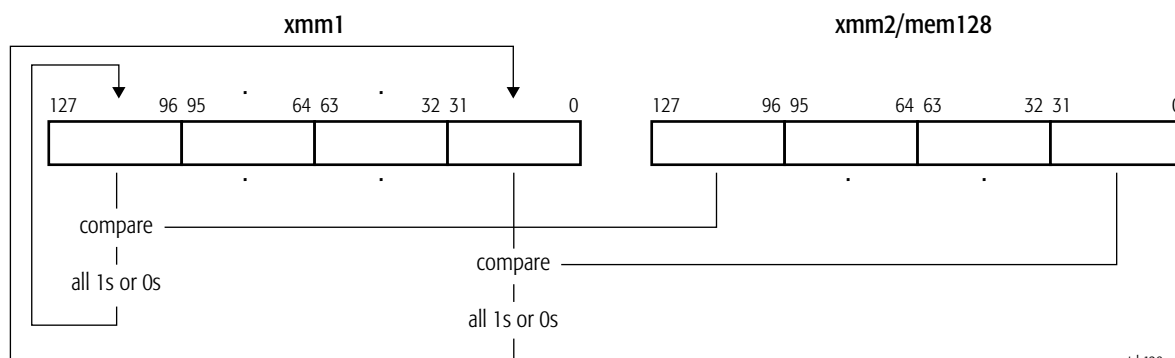
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPGTD**Packed Compare Greater Than Signed Doublewords**

The PCMPGTD instruction compares corresponding packed signed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPGTD <i>xmm1, xmm2/mem128</i>	66 0F 66/r	Compares packed signed 32-bit values in an XMM register and an XMM register or 128-bit memory location.



pcmpgtd-128.eps

Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

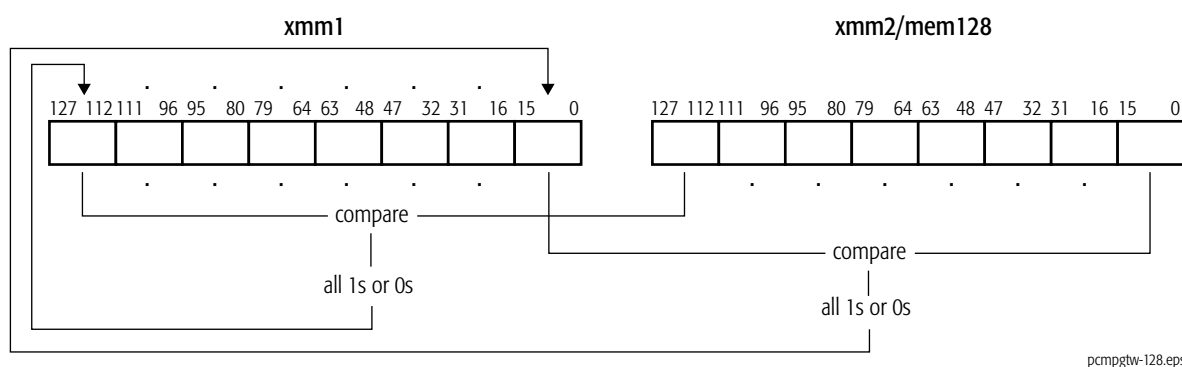
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PCMPGTW**Packed Compare Greater Than Signed Words**

The PCMPGTW instruction compares corresponding packed signed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PCMPGTW <i>xmm1, xmm2/mem128</i>	66 0F 65 /r	Compares packed signed 16-bit values in an XMM register and an XMM register or 128-bit memory location.



pcmpgtw-128.eps

Related Instructions

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PEXTRW Extract Packed Word

The PEXTRW instruction extracts a 16-bit value from an XMM register, as selected by the immediate byte operand (as shown in Table 2-6) and writes it to the low-order word of a 32-bit or 64-bit general-purpose register, with zero-extension to 32 or 64 bits.

Mnemonic	Opcode	Description
PEXTRW <i>reg32/64, xmm, imm8</i>	66 0F C5 /r <i>ib</i>	Extracts a 16-bit value from an XMM register and writes it to low-order 16 bits of a general-purpose register.

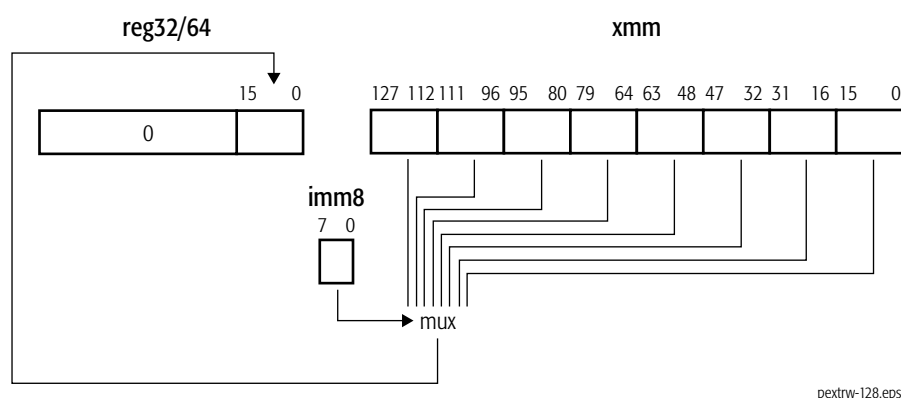


Table 2-6. Immediate-Byte Operand Encoding for 128-Bit PEXTRW

Immediate-Byte Bit Field	Value of Bit Field	Source Bits Extracted
2:0	0	15:0
	1	31:16
	2	47:32
	3	63:48
	4	79:64
	5	95:80
	6	111:96
	7	127:112

Related Instructions

PINSRW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

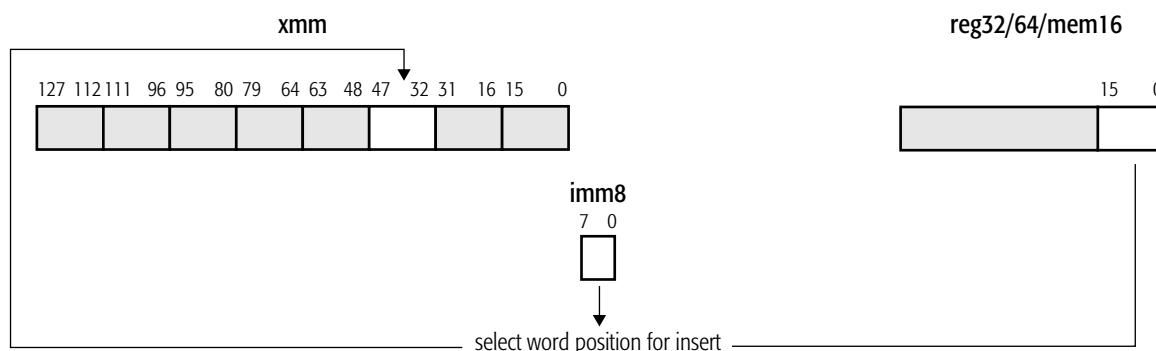
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PINSRW Packed Insert Word

The PINSRW instruction inserts a 16-bit value from the low-order word of a 32-bit or 64-bit general purpose register or a 16-bit memory location into an XMM register. The location in the destination register is selected by the immediate byte operand, as shown in Table 2-7. The other words in the destination register operand are not modified.

Mnemonic	Opcode	Description
PINSRW <i>xmm, reg32/64/mem16, imm8</i>	66 0F C4 /r <i>ib</i>	Inserts a 16-bit value from a general-purpose register or memory location into an XMM register.



pinsrw-128.eps

Table 2-7. Immediate-Byte Operand Encoding for 128-Bit PINSRW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled
2:0	0	15:0
	1	31:16
	2	47:32
	3	63:48
	4	79:64
	5	95:80
	6	111:96
	7	127:112

Related Instructions

PEXTRW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

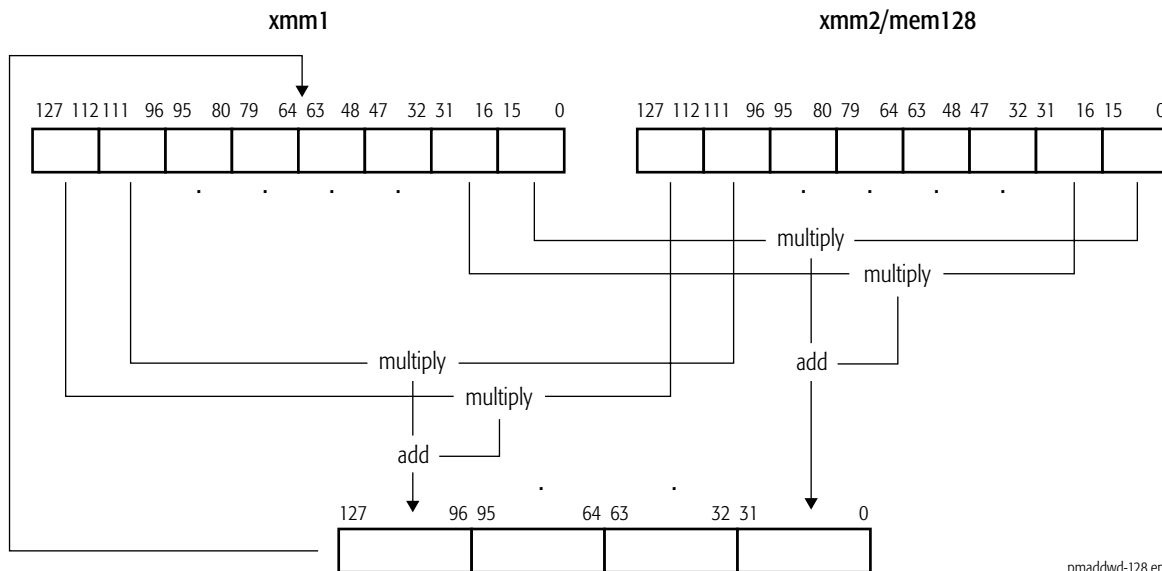
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMADDWD**Packed Multiply Words and Add Doublewords**

The PMADDWD instruction multiplies each packed 16-bit signed value in the first source operand by the corresponding packed 16-bit signed value in the second source operand, adds the adjacent intermediate 32-bit results of each multiplication (for example, the multiplication results for the adjacent bit fields 63:48 and 47:32, and 31:16 and 15:0), and writes the 32-bit result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMADDWD <i>xmm1, xmm2/mem128</i>	66 0F F5 /r	Multiplies eight packed 16-bit signed values in an XMM register and another XMM register or 128-bit memory location, adds intermediate results, and writes the result in the destination XMM register.



If all four of the 16-bit source operands used to produce a 32-bit multiply-add result have the value 8000h, the 32-bit result is 8000_0000h, which is incorrect.

Related Instructions

PMULHUW, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

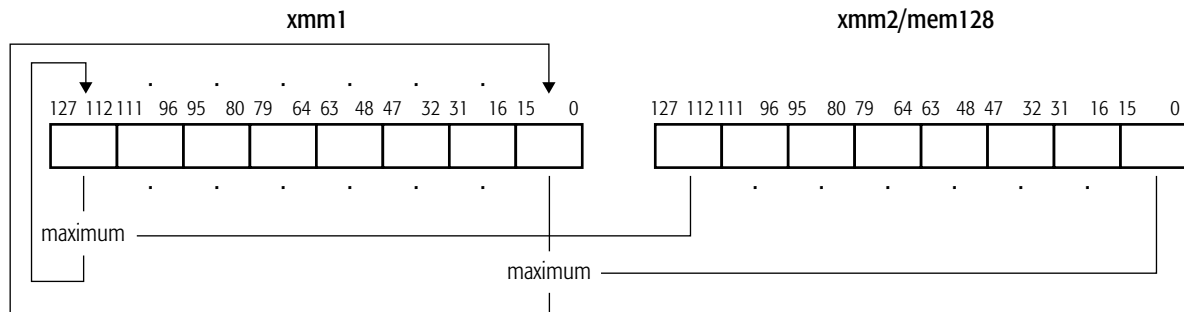
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMAXSW**Packed Maximum Signed Words**

The PMAXSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an XMM register and an XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMAXSW <i>xmm1, xmm2/mem128</i>	66 0F EE /r	Compares packed signed 16-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the maximum value of each compare in destination XMM register.



pmaxsw-128.eps

Related Instructions

PMAXUB, PMINSW, PMINUB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

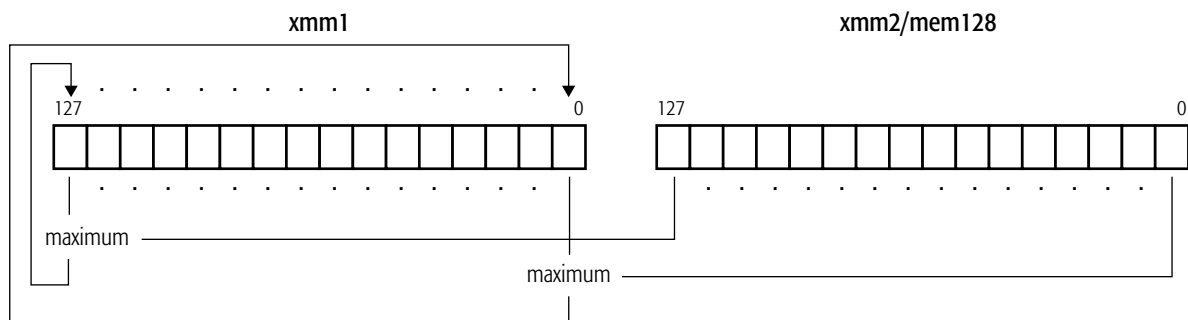
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	A memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMAXUB**Packed Maximum Unsigned Bytes**

The PMAXUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination and second source operands are an XMM register and an XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMAXUB <i>xmm1, xmm2/mem128</i>	66 0F DE /r	Compares packed unsigned 8-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the maximum value of each compare in the destination XMM register.



pmaxub-128.eps

Related Instructions

PMAXSW, PMINSW, PMINUB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

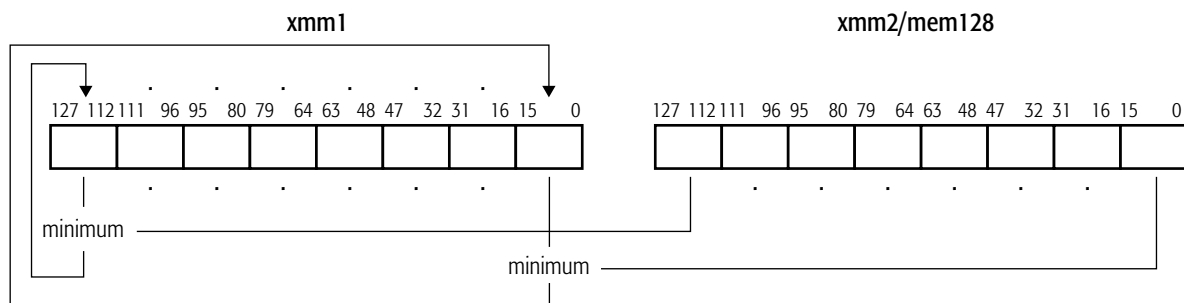
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	A memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMINSW**Packed Minimum Signed Words**

The PMINSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an XMM register and an XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMINSW <i>xmm1, xmm2/mem128</i>	66 0F EA /r	Compares packed signed 16-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the minimum value of each compare in the destination XMM register.



pminsw-128.eps

Related Instructions

PMAXSW, PMAXUB, PMINUB

rFLAGS Affected

None.

MXCSR Flags Affected

None.

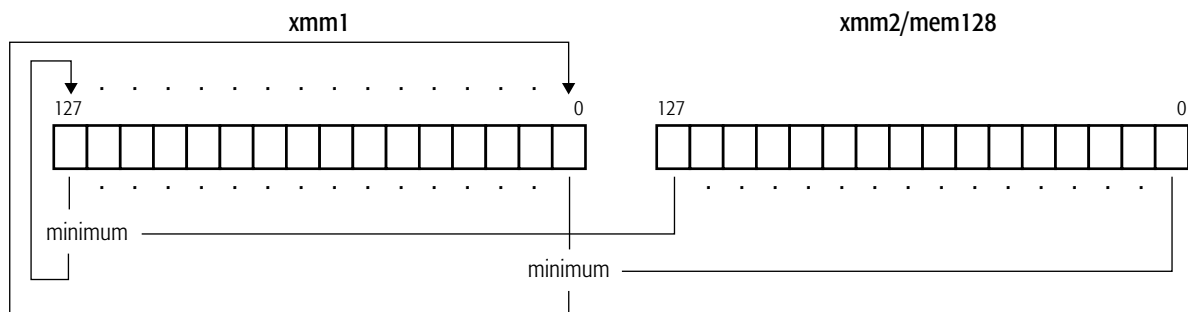
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	A memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMINUB**Packed Minimum Unsigned Bytes**

The PMINUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMINUB <i>xmm1, xmm2/mem64</i>	66 0F DA/ <i>r</i>	Compares packed unsigned 8-bit integer values in an XMM register and another XMM register or 128-bit memory location and writes the minimum value of each comparison in the destination XMM register.



pminub-128.eps

Related Instructions

PMAXSW, PMAXUB, PMINSW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

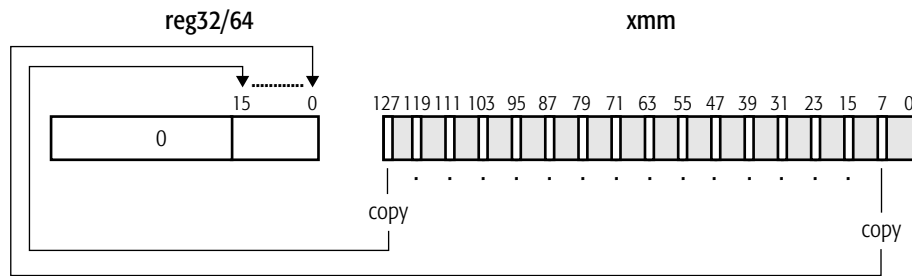
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	A memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMOVMSKB Packed Move Mask Byte

The PMOVMSKB instruction moves the most-significant bit of each byte in the source operand to the destination, with zero-extension to 32 or 64 bits. The destination and source operands are a 32-bit or 64-bit general-purpose register and an XMM register. The result is written to the low-order word of the general-purpose register.

Mnemonic	Opcode	Description
PMOVMSKB <i>reg32/64, xmm</i>	66 0F D7 /r	Moves most-significant bit of each byte in an XMM register to low-order word of a 32-bit or 64-bit general-purpose register.



pmovmskb-128.eps

Related Instructions

MOVMSKPD, MOVMSKPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

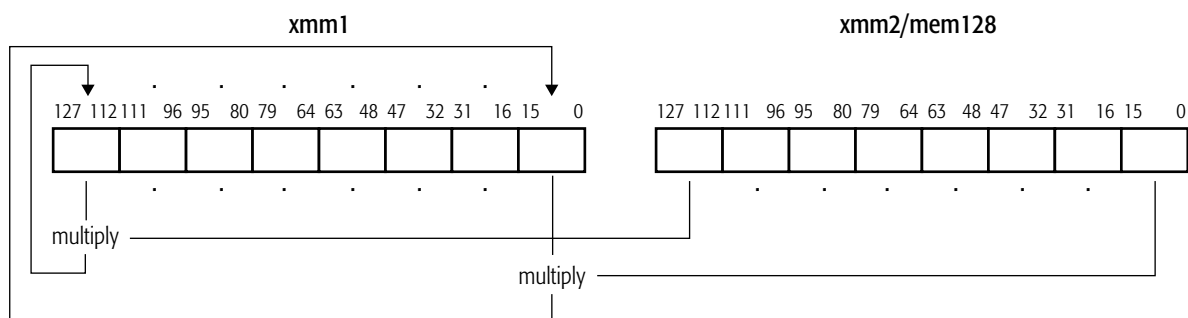
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

PMULHUW**Packed Multiply High Unsigned Word**

The PMULHUW instruction multiplies each packed unsigned 16-bit values in the first source operand by the corresponding packed unsigned word in the second source operand and writes the high-order 16 bits of each intermediate 32-bit result in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMULHUW <i>xmm1, xmm2/mem128</i>	66 0F E4 /r	Multiplies packed 16-bit values in an XMM register by the packed 16-bit values in another XMM register or 128-bit memory location and writes the high-order 16 bits of each result in the destination XMM register.



pmulhuw-128.eps

Related Instructions

PMADDWD, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

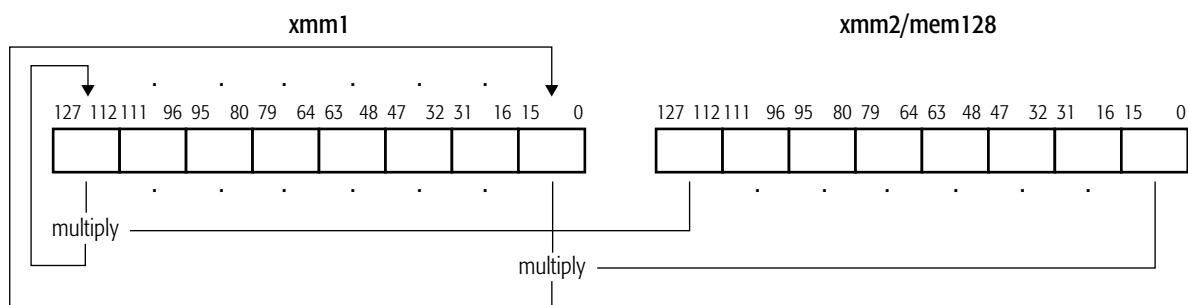
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMULHW**Packed Multiply High Signed Word**

The PMULHW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the high-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMULHW <i>xmm1, xmm2/mem128</i>	66 0F E5 /r	Multiplies packed 16-bit signed integer values in an XMM register and another XMM register or 128-bit memory location and writes the high-order 16 bits of each result in the destination XMM register.



pmulhw-128.eps

Related Instructions

PMADDWD, PMULHUW, PMULLW, PMULUDQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

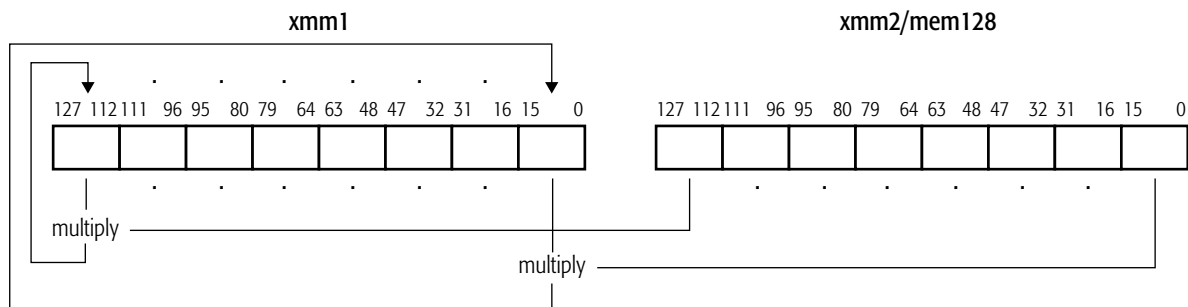
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMULLW**Packed Multiply Low Signed Word**

The PMULLW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the low-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PMULLW <i>xmm1, xmm2/mem128</i>	66 0F D5 /r	Multiplies packed 16-bit signed integer values in an XMM register and another XMM register or 128-bit memory location and writes the low-order 16 bits of each result in the destination XMM register.



pmullw-128.eps

Related Instructions

PMADDWD, PMULHUW, PMULHW, PMULUDQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

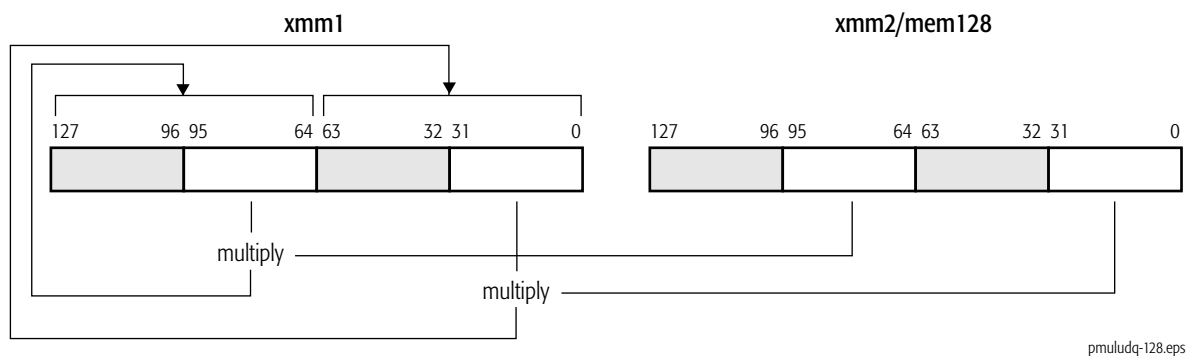
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PMULUDQ**Packed Multiply Unsigned Doubleword and Store Quadword**

The PMULUDQ instruction multiplies two pairs of 32-bit unsigned integer values in the first and second source operands and writes the two 64-bit results in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The source operands are in the first (low-order) and third doublewords of the source operands, and the result of each multiply is stored in the first and second quadwords of the destination XMM register.

Mnemonic	Opcode	Description
PMULUDQ <i>xmm1, xmm2/mem128</i>	66 0F F4 /r	Multiplies two pairs of 32-bit unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes the two 64-bit results in the destination XMM register.

**Related Instructions**

PMADDWD, PMULHUW, PMULHW, PMULLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

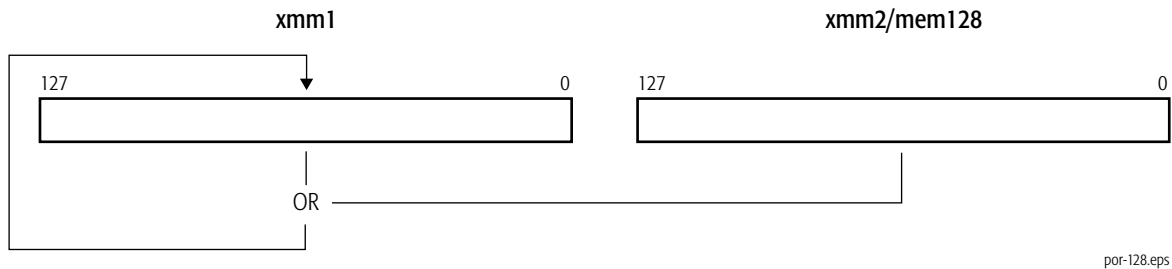
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

POR Packed Logical Bitwise OR

The POR instruction performs a bitwise logical OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
POR <i>xmm1, xmm2/mem128</i>	66 0F EB /r	Performs bitwise logical OR of values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



Related Instructions

PAND, PANDN, PXOR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

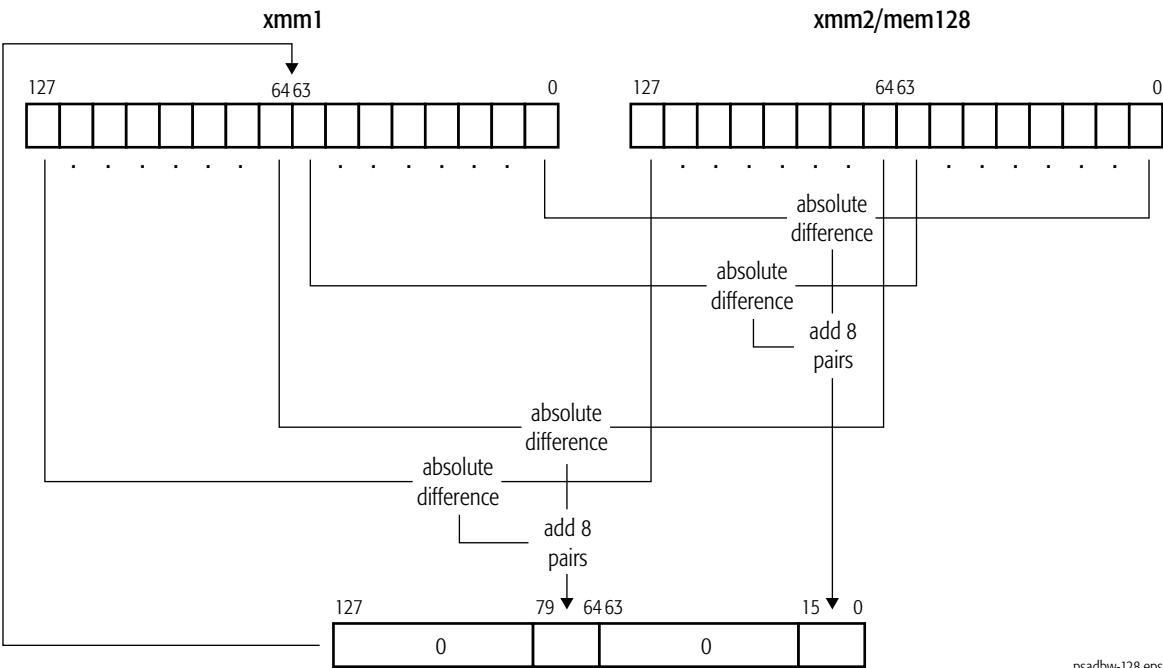
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSADBW

Packed Sum of Absolute Differences of Bytes Into a Word

The PSADBW instruction computes the absolute differences of eight corresponding packed 8-bit unsigned integers in the first and second source operands and writes the unsigned 16-bit integer result of the sum of the eight differences in a word in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSADBW <i>xmm1, xmm2/mem128</i>	66 0F F6/r	Compute the sum of the absolute differences of two sets of packed 8-bit unsigned integer values in an XMM register and another XMM register or 128-bit memory location and writes the 16-bit unsigned integer result in the destination XMM register.



psadbw-128.eps

The sum of the differences of the eight bytes in the high-order quadwords of the source operands are written in the least-significant word of the high-order quadword in the destination XMM register, with the remaining bytes cleared to all 0s. The sum of the differences of the eight bytes in the low-order quadwords of the source operands

are written in the least-significant word of the low-order quadword in the destination XMM register, with the remaining bytes cleared to all 0s.

rFLAGS Affected

None.

MXCSR Flags Affected

None.

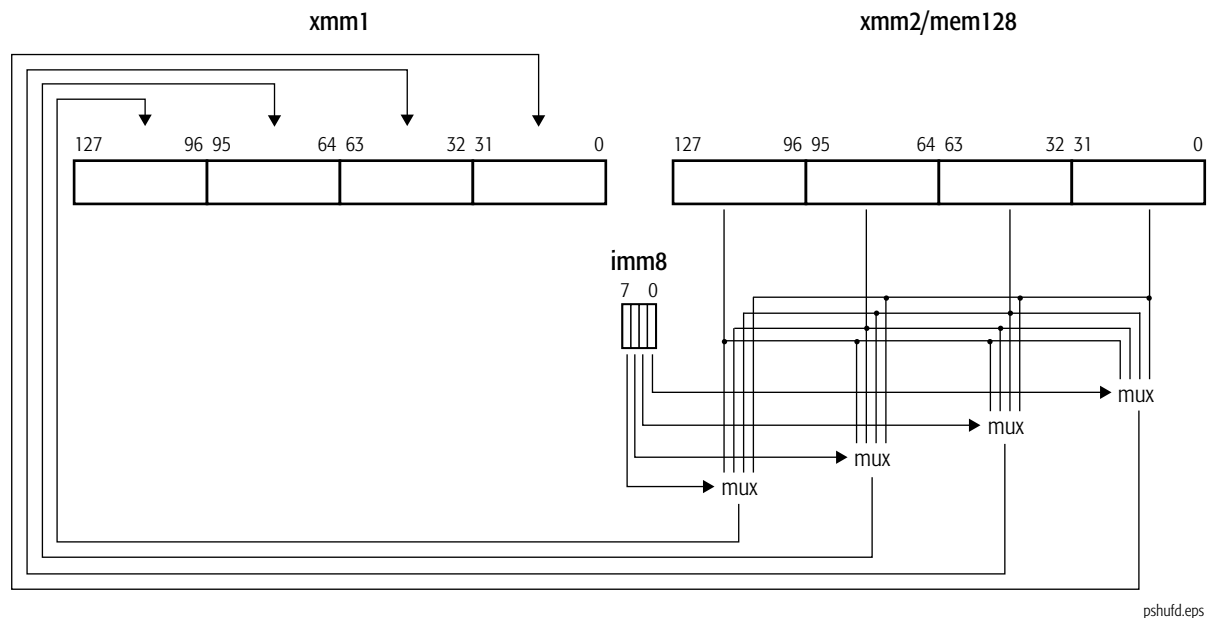
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSHUFD Packed Shuffle Doublewords

The PSHUFD instruction moves any one of the four packed doublewords in an XMM register or 128-bit memory location to each doubleword in another XMM register. In each case, the value of the destination doubleword is determined by a two-bit field in the immediate-byte operand, with bits 0 and 1 selecting the contents of the low-order doubleword, bits 2 and 3 selecting the second doubleword, bits 4 and 5 selecting the third doubleword, and bits 6 and 7 selecting the high-order doubleword. Refer to Table 2-8. A doubleword in the source operand may be copied to more than one doubleword in the destination.

Mnemonic	Opcode	Description
PSHUFD <i>xmm1, xmm2/mem128, imm8</i>	66 0F 70 /r <i>ib</i>	Moves packed 32-bit values in an XMM register or 128-bit memory location to doubleword locations in another XMM register, as selected by the immediate-byte operand.



pshufd.eps

Table 2-8. Immediate-Byte Operand Encoding for PSHUFD

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source Bits Moved
1:0	0	31:0	31:0
	1	31:0	63:32
	2	31:0	95:64
	3	31:0	127:96
3:2	0	63:32	31:0
	1	63:32	63:32
	2	63:32	95:64
	3	63:32	127:96
5:4	0	95:64	31:0
	1	95:64	63:32
	2	95:64	95:64
	3	95:64	127:96
7:6	0	127:96	31:0
	1	127:96	63:32
	2	127:96	95:64
	3	127:96	127:96

Related Instructions

PSHUFHW, PSFUFLW, PSFUFW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

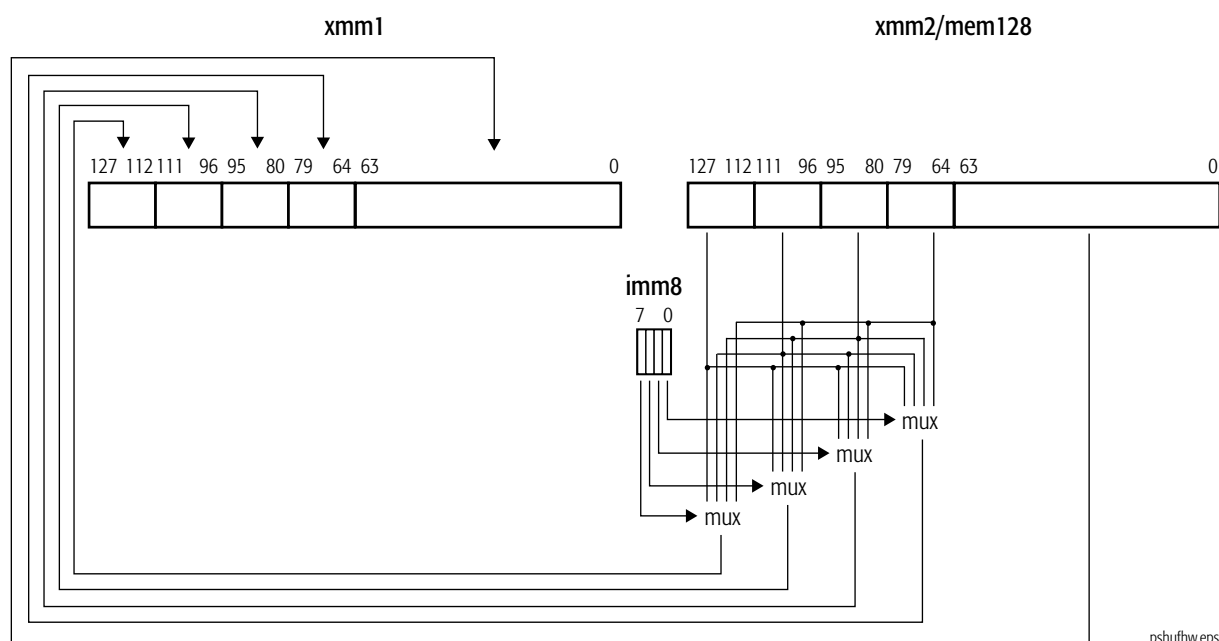
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSHUFHW**Packed Shuffle High Words**

The PSHUFHW instruction moves any one of the four packed words in the high-order quadword of an XMM register or 128-bit memory location to each word in the high-order quadword of another XMM register. In each case, the value of the destination word is determined by a two-bit field in the immediate-byte operand, with bits 0 and 1 selecting the contents of the low-order word, bits 2 and 3 selecting the second word, bits 4 and 5 selecting the third word, and bits 6 and 7 selecting the high-order word. Refer to Table 2-9. A word in the source operand may be copied to more than one word in the destination. The low-order quadword of the source operand is copied to the low-order quadword of the destination register.

Mnemonic	Opcode	Description
PSHUFHW <i>xmm1, xmm2/mem128, imm8</i>	F3 0F 70 /r ib	Shuffles packed 16-bit values in high-order quadword of an XMM register or 128-bit memory location and puts the result in high-order quadword of another XMM register.



pshufhw.eps

Table 2-9. Immediate-Byte Operand Encoding for PSHUFW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source Bits Moved
1:0	0	79:64	79:64
	1	79:64	95:80
	2	79:64	111:96
	3	79:64	127:112
3:2	0	95:80	79:64
	1	95:80	95:80
	2	95:80	111:96
	3	95:80	127:112
5:4	0	111:96	79:64
	1	111:96	95:80
	2	111:96	111:96
	3	111:96	127:112
7:6	0	127:112	79:64
	1	127:112	95:80
	2	127:112	111:96
	3	127:112	127:112

Related Instructions

PSHUFD, PSHUFLW, PSHUFW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSHUFLW**Packed Shuffle Low Words**

The PSHUFLW instruction moves any one of the four packed words in the low-order quadword of an XMM register or 128-bit memory location to each word in the low-order quadword of another XMM register. In each case, the selection of the value of the destination word is determined by a two-bit field in the immediate-byte operand, with bits 0 and 1 selecting the contents of the low-order word, bits 2 and 3 selecting the second word, bits 4 and 5 selecting the third word, and bits 6 and 7 selecting the high-order word. Refer to Table 2-10. A word in the source operand may be copied to more than one word in the destination. The high-order quadword of the source operand is copied to the high-order quadword of the destination register.

Mnemonic	Opcode	Description
PSHUFLW <i>xmm1, xmm2/mem128, imm8</i>	F2 0F 70 /r <i>ib</i>	Shuffles packed 16-bit values in low-order quadword of an XMM register or 128-bit memory location and puts the result in low-order quadword of another XMM register.

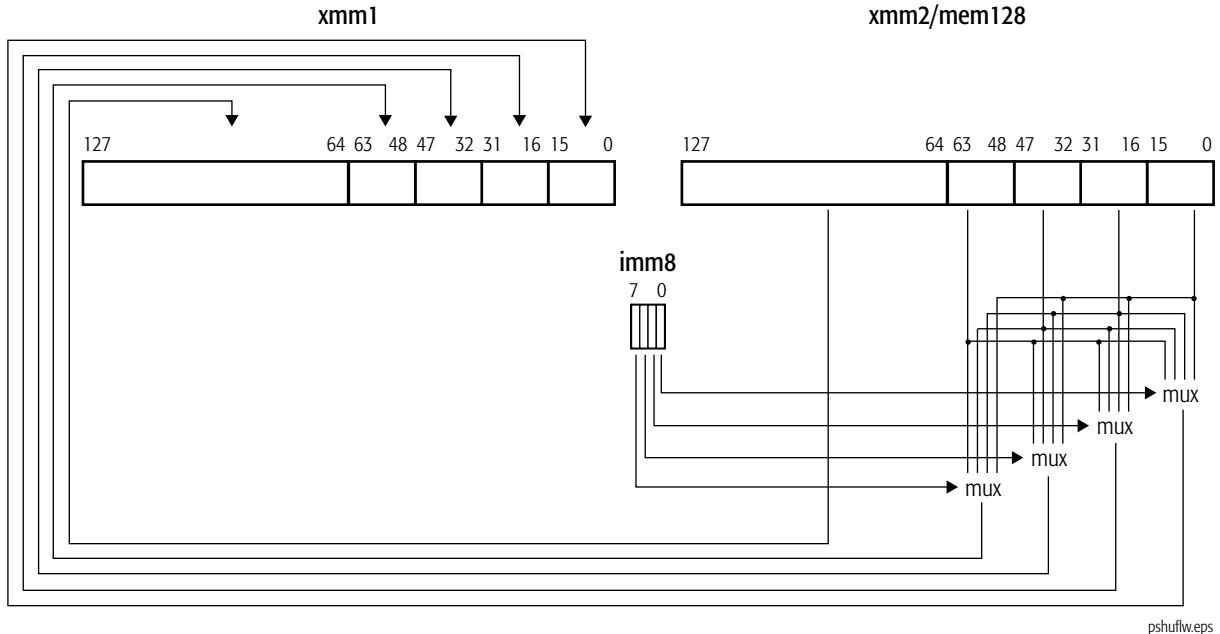


Table 2-10. Immediate-Byte Operand Encoding for PSHUFLW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source Bits Moved
1:0	0	15:0	15:0
	1	15:0	31:16
	2	15:0	47:32
	3	15:0	63:48
3:2	0	31:16	15:0
	1	31:16	31:16
	2	31:16	47:32
	3	31:16	63:48
5:4	0	47:32	15:0
	1	47:32	31:16
	2	47:32	47:32
	3	47:32	63:48
7:6	0	63:48	15:0
	1	63:48	31:16
	2	63:48	47:32
	3	63:48	63:48

Related Instructions

PSHUFD, PSHUFHW, PSHUFW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

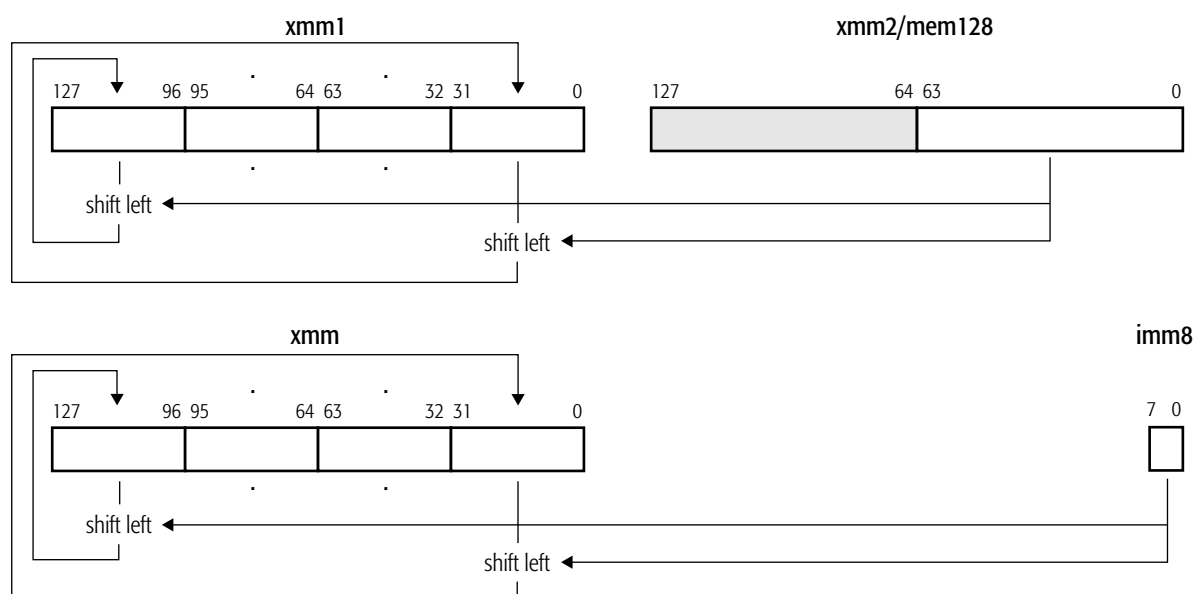
PSLLD**Packed Shift Left Logical Doublewords**

The PSLLD instruction left-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLD <i>xmm1, xmm2/mem128</i>	66 0F F2 /r	Left-shifts packed doublewords in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSLLD <i>xmm, imm8</i>	66 0F 72 /6 ib	Left-shifts packed doublewords in an XMM register by the amount specified in an immediate byte value.



pslld-128.eps

Related Instructions

PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

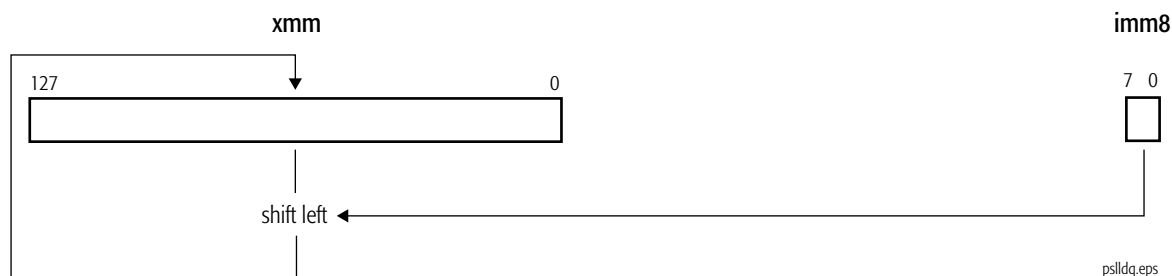
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSLLDQ**Packed Shift Left Logical Double Quadword**

The PSLLDQ left-shifts the 128-bit (double quadword) value in an XMM register by the number of bytes specified in an immediate byte value. The low-order bytes that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination XMM register is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLDQ <i>xmm, imm8</i>	66 0F 73 /7 <i>ib</i>	Left-shifts double quadword value in an XMM register by the amount specified in an immediate byte value.

**Related Instructions**

PSLLD, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

PSLLQ

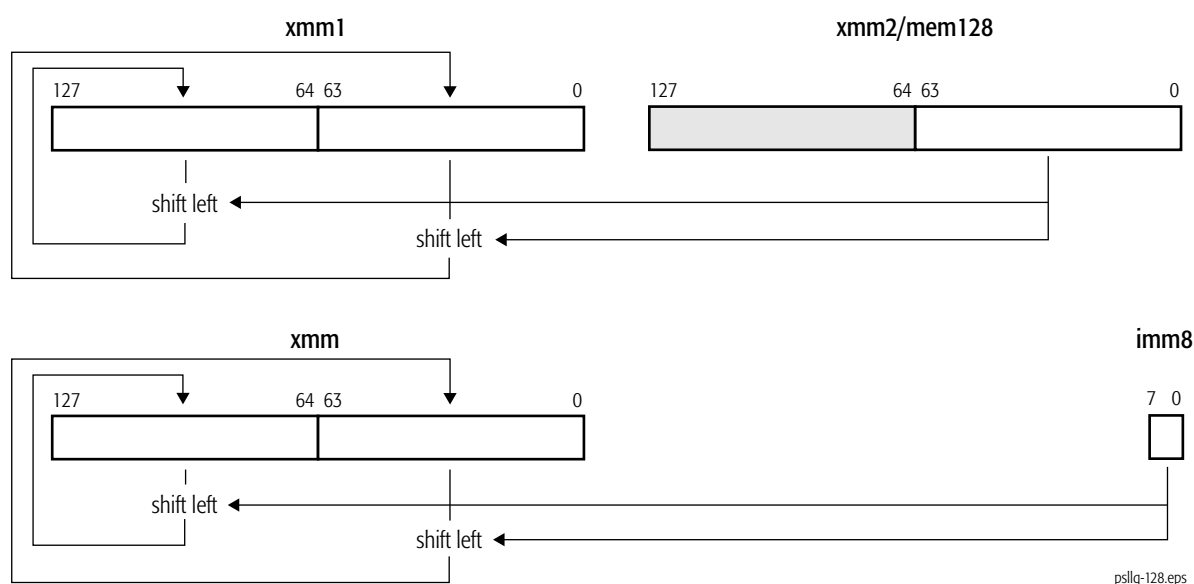
Packed Shift Left Logical Quadwords

The PSLLQ instruction left-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLQ <i>xmm1, xmm2/mem128</i>	66 0F F3 /r	Left-shifts packed quadwords in XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSLLQ <i>xmm, imm8</i>	66 0F 73 /6 ib	Left-shifts packed quadwords in an XMM register by the amount specified in an immediate byte value.



psllq-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

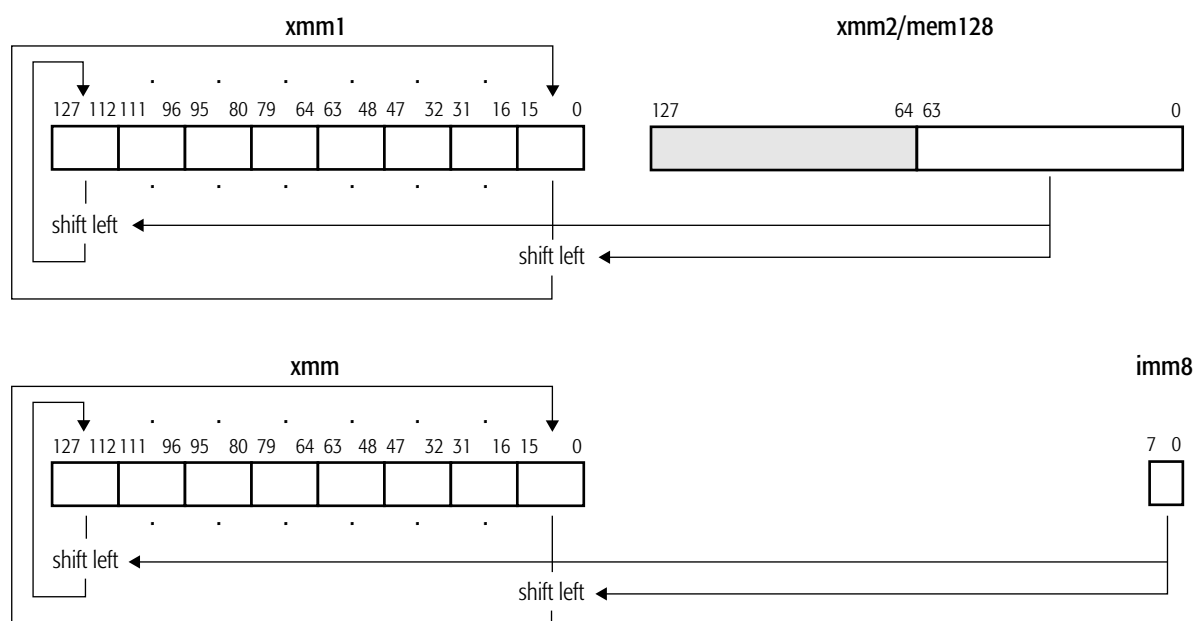
PSLLW**Packed Shift Left Logical Words**

The PSLLW instruction left-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLW <i>xmm1, xmm2/mem128</i>	66 0F F1 / <i>r</i>	Left-shifts packed words in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSLLW <i>xmm, imm8</i>	66 0F 71 / <i>6 ib</i>	Left-shifts packed words in an XMM register by the amount specified in an immediate byte value.



psllw-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

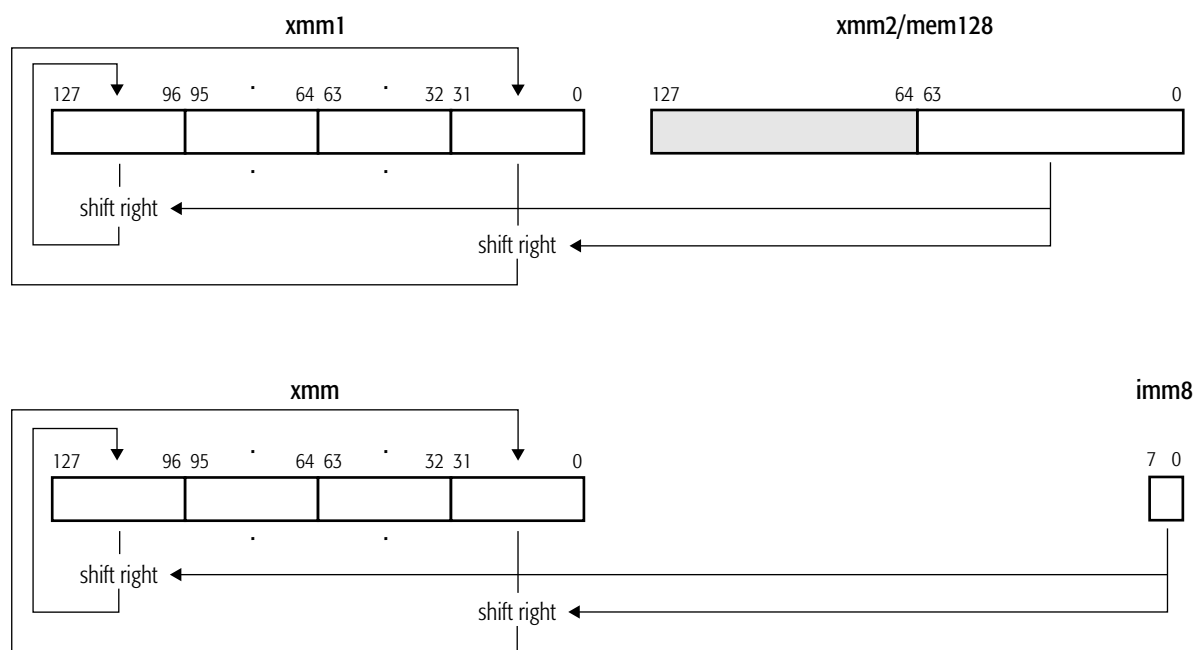
PSRAD**Packed Shift Right Arithmetic Doublewords**

The PSRAD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the doubleword's initial value. If the shift value is greater than 31, each doubleword in the destination is filled with the sign bit of the doubleword's initial value.

Mnemonic	Opcode	Description
PSRAD <i>xmm1, xmm2/mem128</i>	66 0F E2 /r	Right-shifts packed doublewords in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSRAD <i>xmm, imm8</i>	66 0F 72 /4 ib	Right-shifts packed doublewords in an XMM register by the amount specified in an immediate byte value.



psrad-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

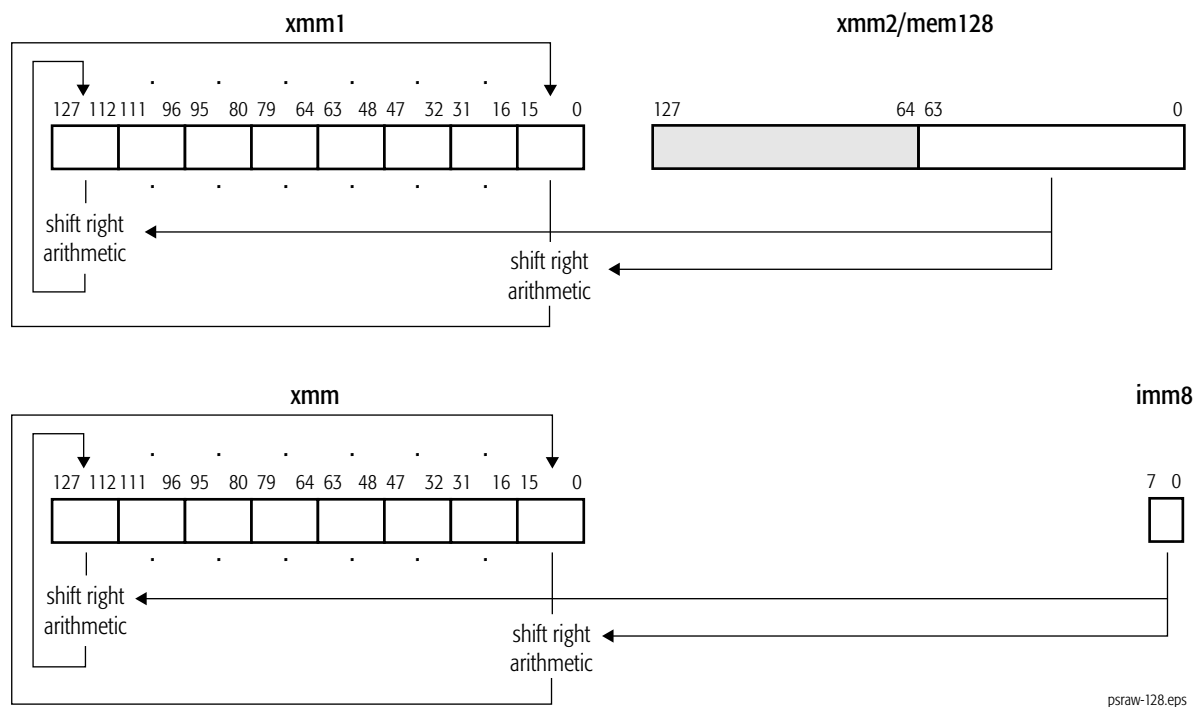
PSRAW**Packed Shift Right Arithmetic Words**

The PSRAW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the word's initial value. If the shift value is greater than 15, each word in the destination is filled with the sign bit of the word's initial value.

Mnemonic	Opcode	Description
PSRAW <i>xmm1, xmm2/mem128</i>	66 0F E1 /r	Right-shifts packed words in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSRAW <i>xmm, imm8</i>	66 0F 71 /4 <i>ib</i>	Right-shifts packed words in an XMM register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

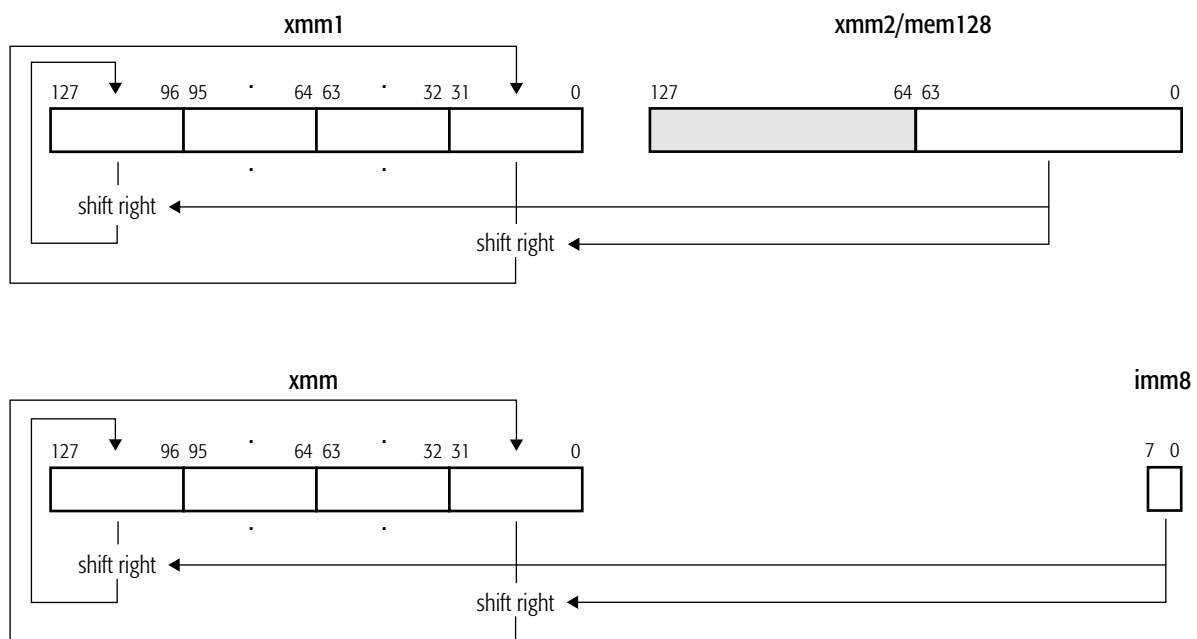
PSRLD**Packed Shift Right Logical Doublewords**

The PSRLD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLD <i>xmm1, xmm2/mem128</i>	66 0F D2 /r	Right-shifts packed doublewords in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSRLD <i>xmm, imm8</i>	66 0F 72 /2 <i>ib</i>	Right-shifts packed doublewords in an XMM register by the amount specified in an immediate byte value.



psrld-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

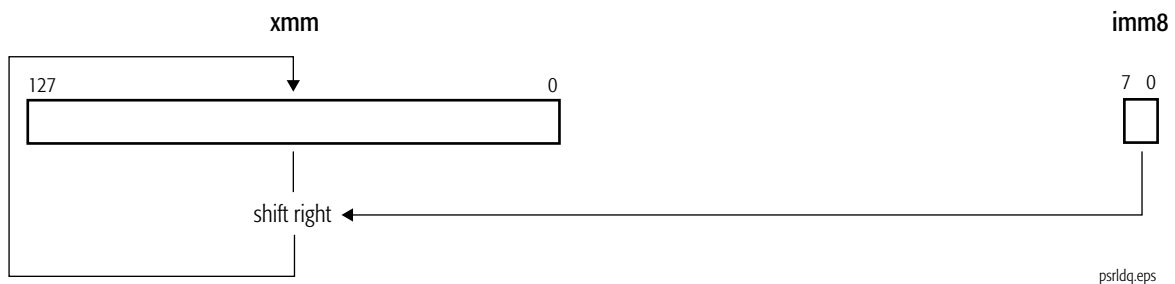
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSRLDQ**Packed Shift Right Logical Double Quadword**

The PSRLDQ right-shifts the 128-bit (double quadword) value in an XMM register by the number of bytes specified in an immediate byte value. The high-order bytes that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination XMM register is cleared to all 0s.

Mnemonic	Opcode	Description
PSRLDQ <i>xmm, imm8</i>	66 0F 73 /3 <i>ib</i>	Right-shifts double quadword value in an XMM register by the amount specified in an immediate byte value.

**Related Instructions**

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.

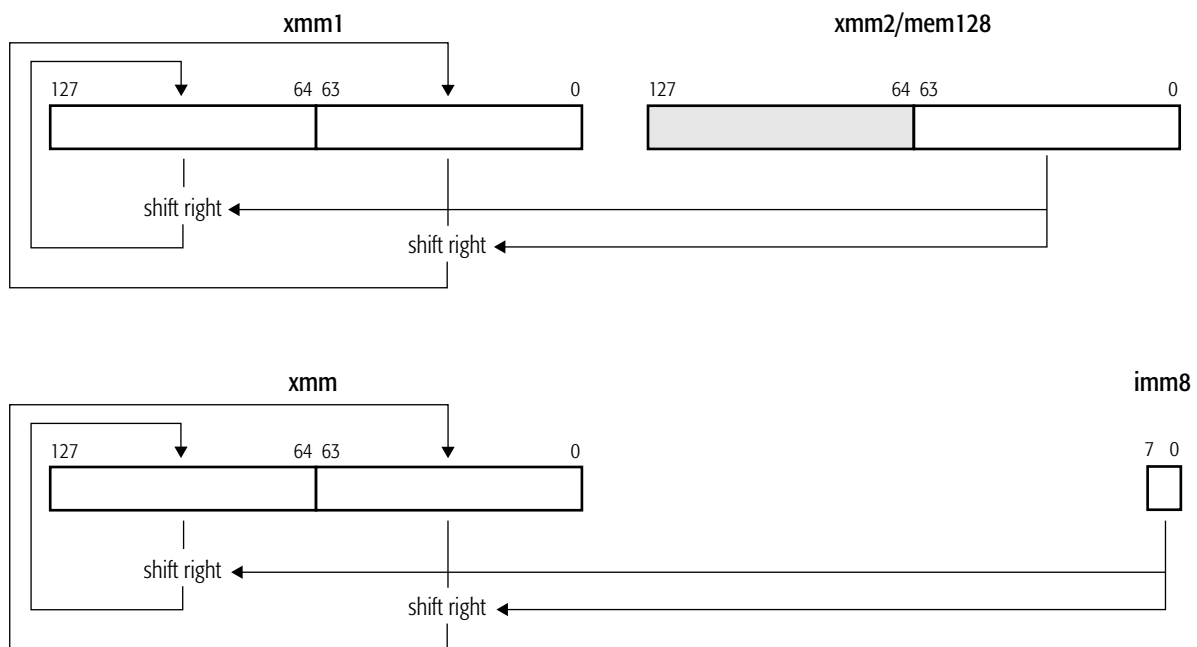
PSRLQ**Packed Shift Right Logical Quadwords**

The PSRLQ instruction right-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLQ <i>xmm1, xmm2/mem128</i>	66 0F D3 /r	Right-shifts packed quadwords in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSRLQ <i>xmm, imm8</i>	66 0F 73 /2 ib	Right-shifts packed quadwords in an XMM register by the amount specified in an immediate byte value.



psrlq-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

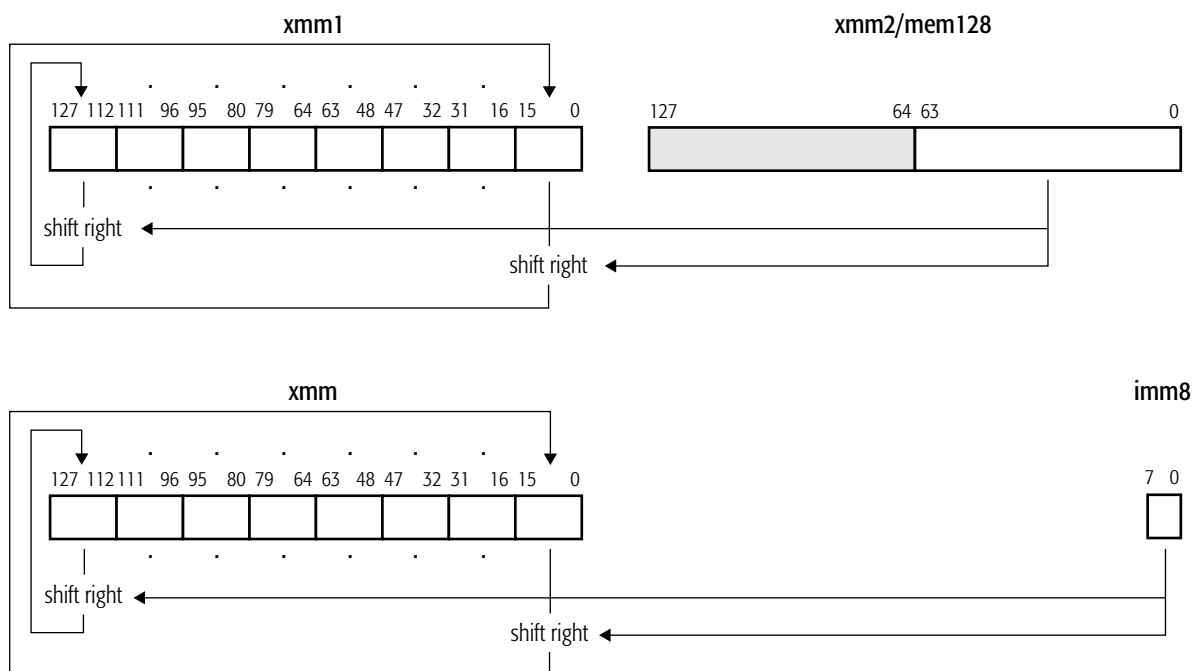
PSRLW**Packed Shift Right Logical Words**

The PSRLW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an XMM register and another XMM register or 128-bit memory location, or
- an XMM register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLW <i>xmm1, xmm2/mem128</i>	66 0F D1 /r	Right-shifts packed words in an XMM register by the amount specified in the low 64 bits of an XMM register or 128-bit memory location.
PSRLW <i>xmm, imm8</i>	66 0F 71 /2 ib	Right-shifts packed words in an XMM register by the amount specified in an immediate byte value.



psrlw-128.eps

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

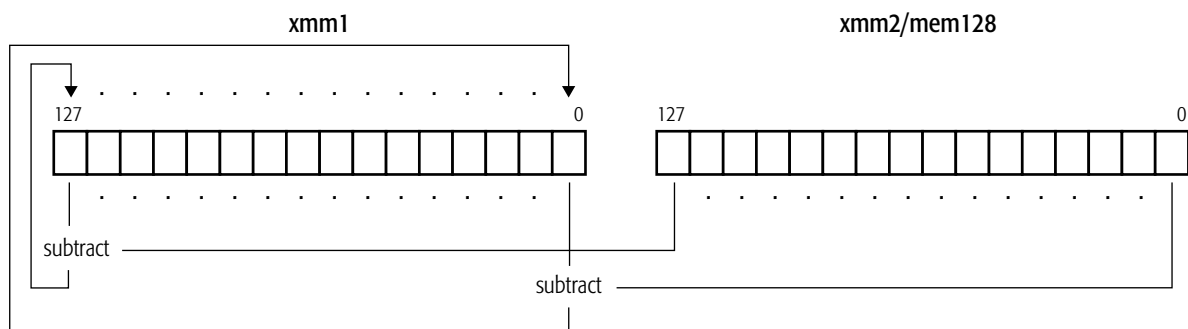
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBB**Packed Subtract Bytes**

The PSUBB instruction subtracts each packed 8-bit integer value in the second source operand from the corresponding packed 8-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBB <i>xmm1, xmm2/mem128</i>	66 0F F8 /r	Subtracts packed byte integer values in an XMM register or 128-bit memory location from packed byte integer values in another XMM register and writes the result in the destination XMM register.



psubb-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

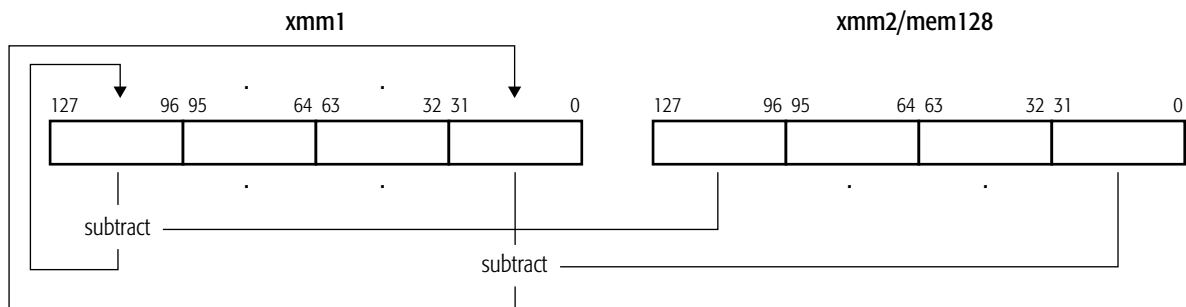
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBD**Packed Subtract Doublewords**

The PSUBD instruction subtracts each packed 32-bit integer value in the second source operand from the corresponding packed 32-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBD <i>xmm1, xmm2/mem128</i>	66 0F FA /r	Subtracts packed 32-bit integer values in an XMM register or 128-bit memory location from packed 32-bit integer values in another XMM register and writes the result in the destination XMM register.



psubd-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

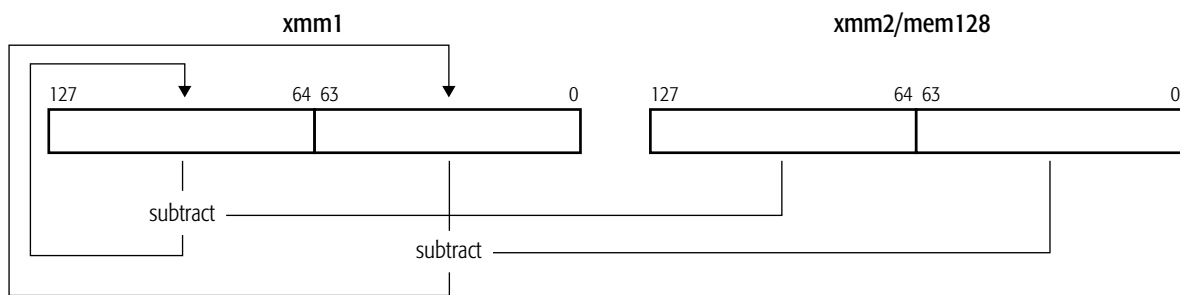
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBQ Packed Subtract Quadword

The PSUBQ instruction subtracts each packed 64-bit integer value in the second source operand from the corresponding packed 64-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination and source operands are an XMM register and another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBQ <i>xmm1, xmm2/mem128</i>	66 0F FB /r	Subtracts packed 64-bit integer values in an XMM register or 128-bit memory location from packed 64-bit integer values in another XMM register and writes the result in the destination XMM register.



psubq-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

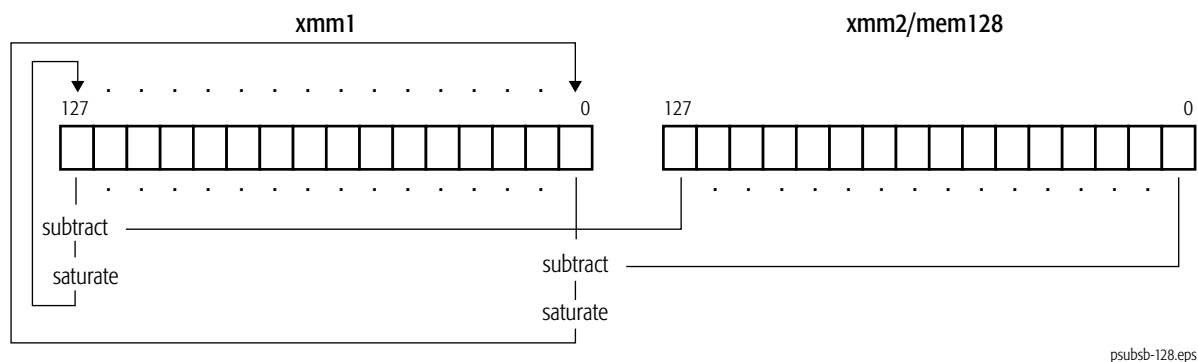
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBSB**Packed Subtract Signed With Saturation Bytes**

The PSUBSB instruction subtracts each packed 8-bit signed integer value in the second source operand from the corresponding packed 8-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBSB <i>xmm1, xmm2/mem128</i>	66 0F E8 /r	Subtracts packed byte signed integer values in an XMM register or 128-bit memory location from packed byte integer values in another XMM register and writes the result in the destination XMM register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

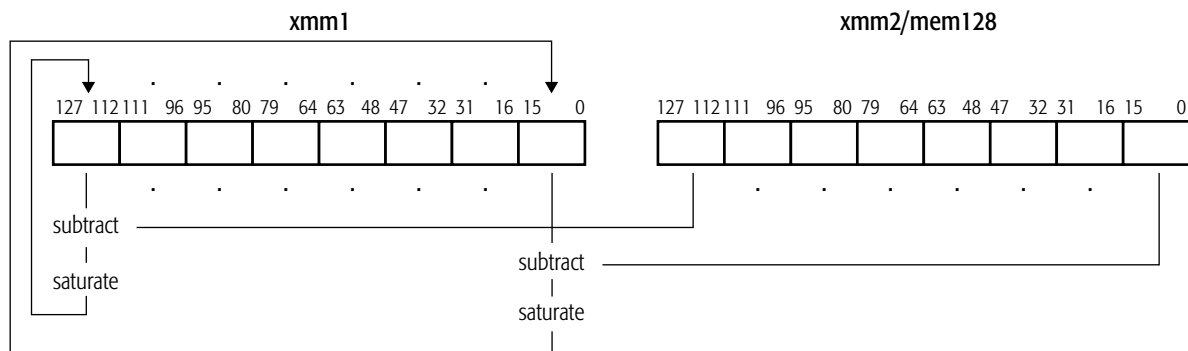
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBSW**Packed Subtract Signed With Saturation Words**

The PSUBSW instruction subtracts each packed 16-bit signed integer value in the second source operand from the corresponding packed 16-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination and source operands are an XMM register and another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBSW <i>xmm1, xmm2/mem128</i>	66 0F E9 /r	Subtracts packed 16-bit signed integer values in an XMM register or 128-bit memory location from packed 16-bit integer values in another XMM register and writes the result in the destination XMM register.



psubsw-128.eps

For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

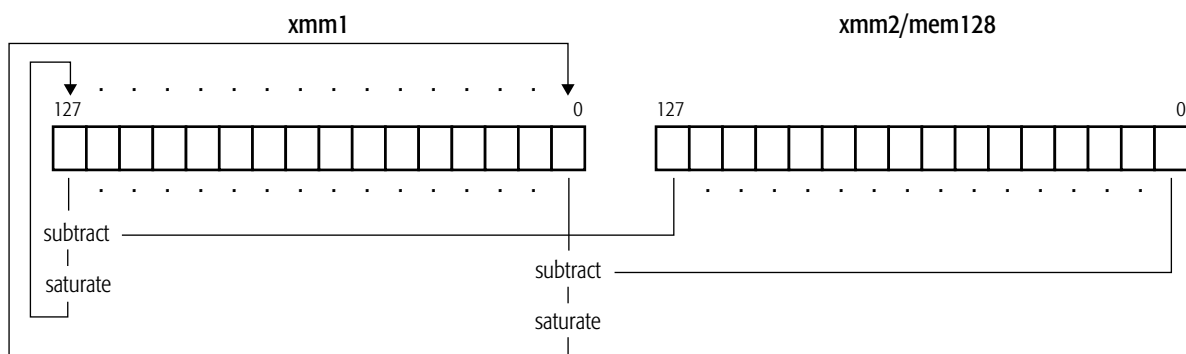
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBUSB**Packed Subtract Unsigned and Saturate Bytes**

The PSUBUSB instruction subtracts each packed 8-bit unsigned integer value in the second source operand from the corresponding packed 8-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBUSB <i>xmm1, xmm2/mem128</i>	66 0F D8 /r	Subtracts packed byte unsigned integer values in an XMM register or 128-bit memory location from packed byte integer values in another XMM register and writes the result in the destination XMM register.



psubusb-128.eps

For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSW, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

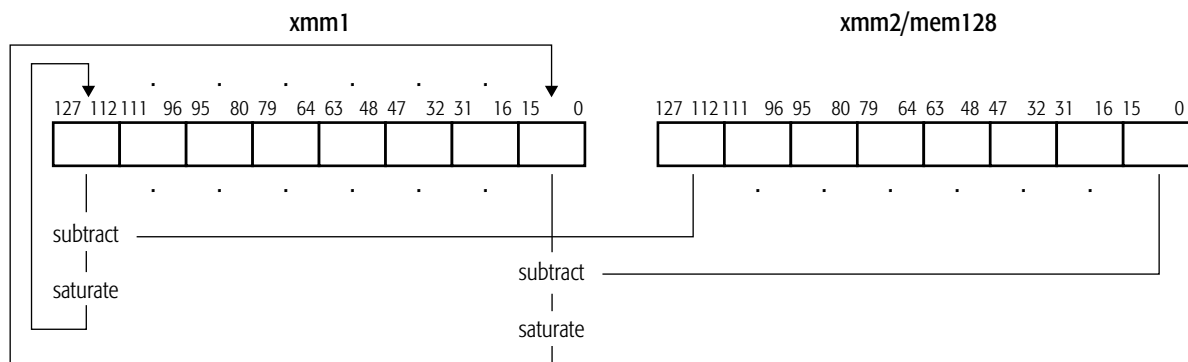
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBUSW**Packed Subtract Unsigned and Saturate Words**

The PSUBUSW instruction subtracts each packed 16-bit unsigned integer value in the second source operand from the corresponding packed 16-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBUSW <i>xmm1</i> , <i>xmm2/mem128</i>	66 0F D9 /r	Subtracts packed 16-bit unsigned integer values in an XMM register or 128-bit memory location from packed 16-bit integer values in another XMM register and writes the result in the destination XMM register.



psubusw-128.eps

For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

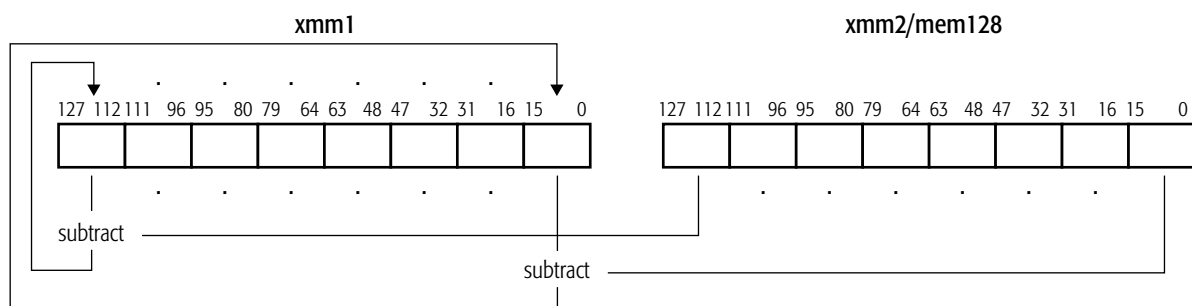
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PSUBW**Packed Subtract Words**

The PSUBW instruction subtracts each packed 16-bit integer value in the second source operand from the corresponding packed 16-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PSUBW <i>xmm1, xmm2/mem128</i>	66 0F F9 /r	Subtracts packed 16-bit integer values in an XMM register or 128-bit memory location from packed 16-bit integer values in another XMM register and writes the result in the destination XMM register.



psubw-128.eps

This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW

rFLAGS Affected

None.

MXCSR Flags Affected

None.

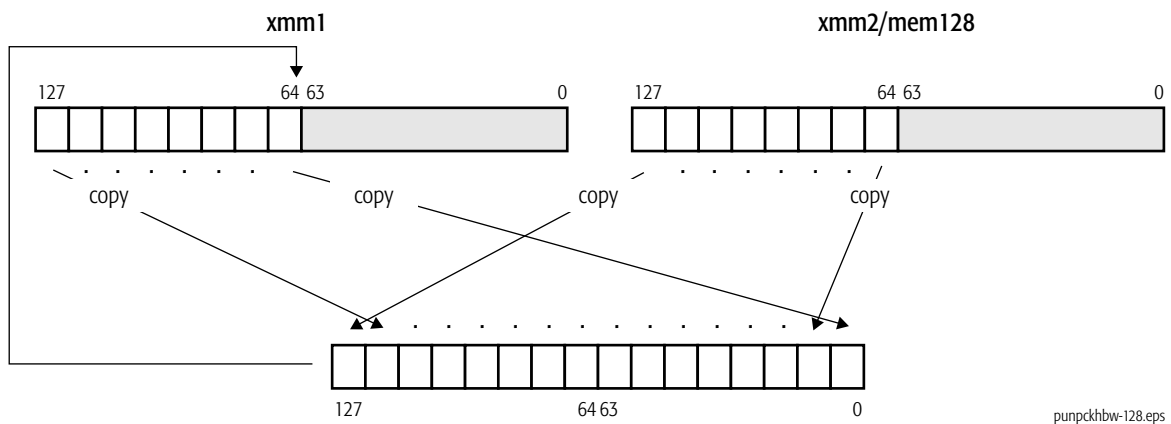
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKHBW**Unpack and Interleave High Bytes**

The PUNPCKHBW instruction unpacks the high-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The low-order bytes of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHBW <i>xmm1, xmm2/mem128</i>	66 0F 68 /r	Unpacks the eight high-order bytes in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved bytes in the destination XMM register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

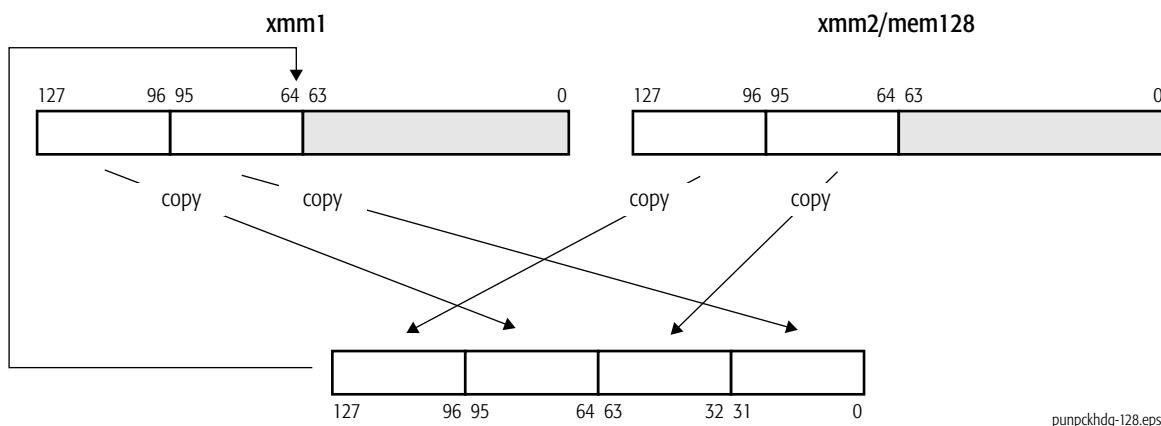
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKHDQ**Unpack and Interleave High Doublewords**

The PUNPCKHDQ instruction unpacks the high-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The low-order doublewords of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHDQ <i>xmm1, xmm2/mem128</i>	66 0F 6A/r	Unpacks two high-order doublewords in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved doublewords in the destination XMM register.



punpckhdq-128.eps

If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

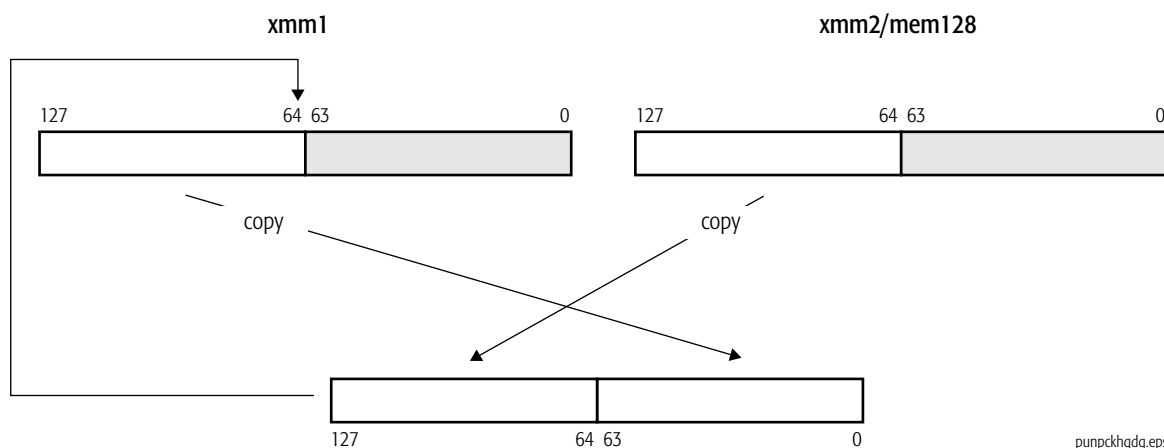
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKHQDQ**Unpack and Interleave High Quadwords**

The PUNPCKHQDQ instruction unpacks the high-order quadwords from the first and second source operands and packs them into interleaved quadwords in the destination (first source). The first source/destination is an XMM register, and the second source operand is another XMM register or 128-bit memory location. The low-order quadwords of the source operands are ignored.

If the second source operand is all 0s, the destination contains the quadword from the first source operand zero-extended to 128 bits. This operation is useful for expanding unsigned 64-bit values to unsigned 128-bit operands for subsequent processing that requires higher precision.

Mnemonic	Opcode	Description
PUNPCKHQDQ <i>xmm1, xmm2/mem128</i>	66 0F 6D /r	Unpacks high-order quadwords in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved quadwords in the destination XMM register.



punpckhqdq.eps

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

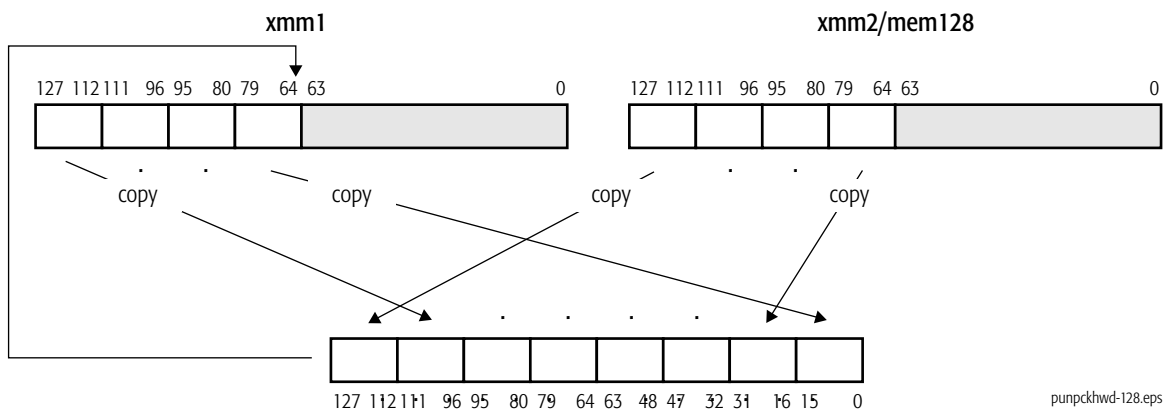
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKHWD**Unpack and Interleave High Words**

The PUNPCKHWD instruction unpacks the high-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The low-order words of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHWD <i>xmm1, xmm2/mem128</i>	66 0F 69 /r	Unpacks four high-order words in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved words in the destination XMM register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

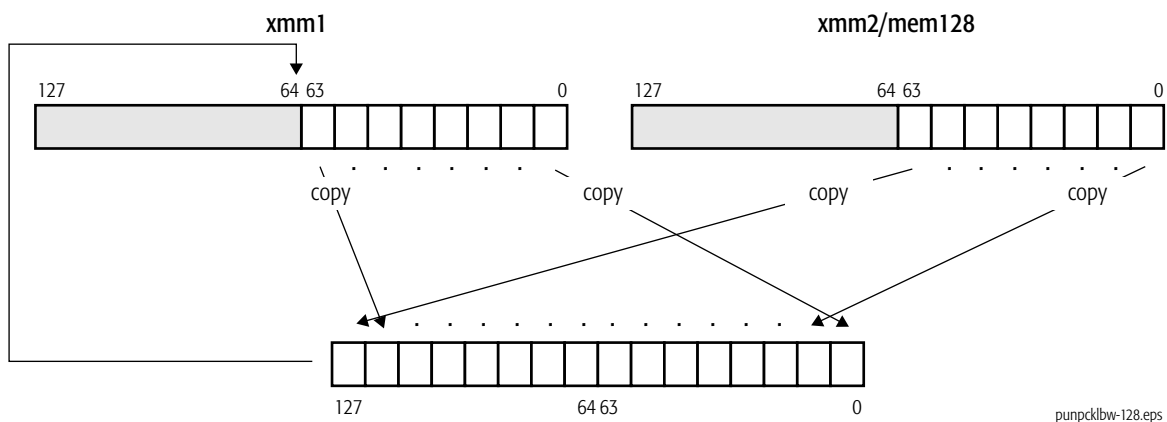
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKLBW**Unpack and Interleave Low Bytes**

The PUNPCKLBW instruction unpacks the low-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The high-order bytes of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLBW <i>xmm1, xmm2/mem128</i>	66 0F 60 /r	Unpacks the eight low-order bytes in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved bytes in the destination XMM register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

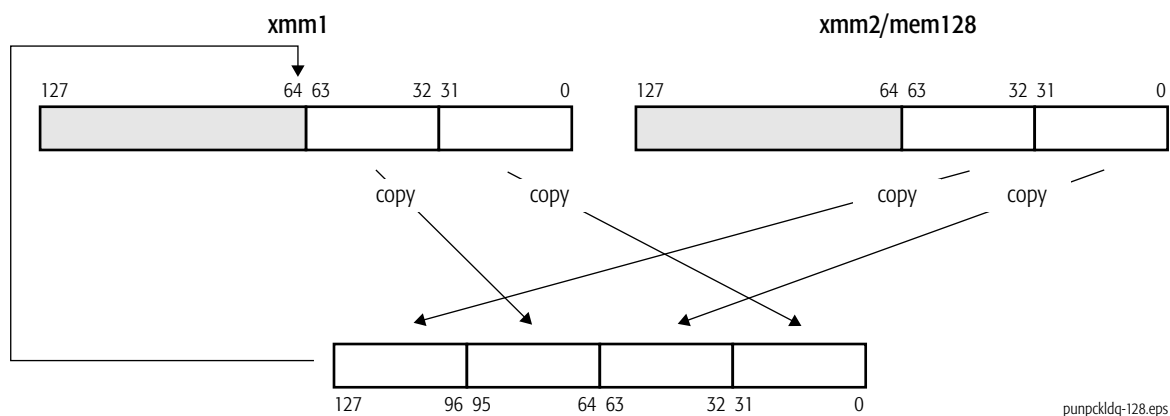
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKLDQ Unpack and Interleave Low Doublewords

The PUNPCKLDQ instruction unpacks the low-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The high-order doublewords of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLDQ <i>xmm1, xmm2/mem128</i>	66 0F 62 /r	Unpacks two low-order doublewords in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved doublewords in the destination XMM register.



punpckldq-128.eps

If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

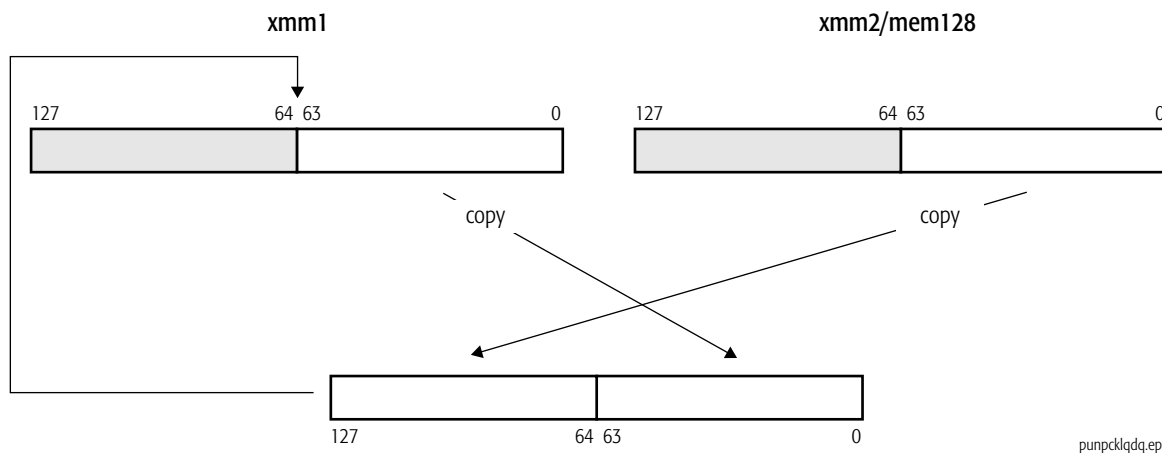
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKLQDQ**Unpack and Interleave Low Quadwords**

The PUNPCKLQDQ instruction unpacks the low-order quadwords from the first and second source operands and packs them into interleaved quadwords in the destination (first source). The first source/destination is an XMM register, and the second source operand is another XMM register or 128-bit memory location. The high-order quadwords of the source operands are ignored.

If the second source operand is all 0s, the destination contains the quadword from the first source operand zero-extended to 128 bits. This operation is useful for expanding unsigned 64-bit values to unsigned 128-bit operands for subsequent processing that requires higher precision.

Mnemonic	Opcode	Description
PUNPCKLQDQ <i>xmm1, xmm2/mem128</i>	66 0F 6C /r	Unpacks low-order quadwords in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved quadwords in the destination XMM register.

**Related Instructions**

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLWD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

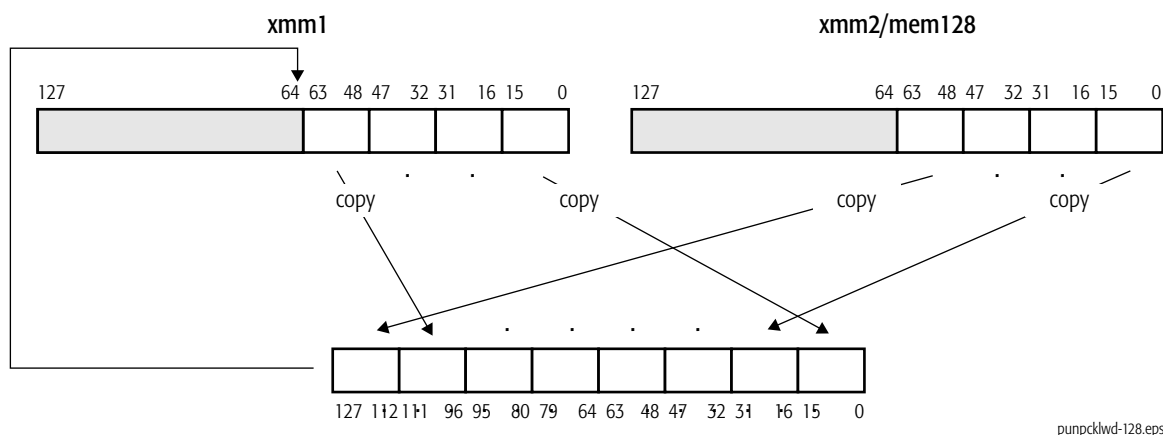
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PUNPCKLWD**Unpack and Interleave Low Words**

The PUNPCKLWD instruction unpacks the low-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The high-order words of the source operands are ignored. The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLWD <i>xmm1, xmm2/mem128</i>	66 0F 61 /r	Unpacks the four low-order words in an XMM register and another XMM register or 128-bit memory location and packs them into interleaved words in the destination XMM register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

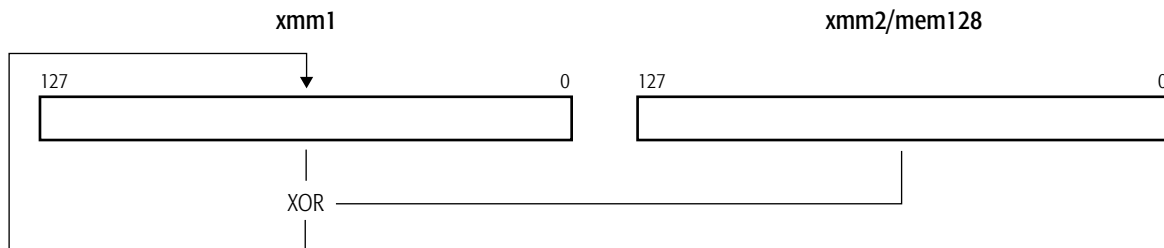
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

PXOR

Packed Logical Bitwise Exclusive OR

The PXOR instruction performs a bitwise exclusive OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an XMM register and the second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
PXOR <i>xmm1, xmm2/mem128</i>	66 0F EF /r	Performs bitwise logical XOR of values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



pxor-128.eps

Related Instructions

PAND, PANDN, POR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

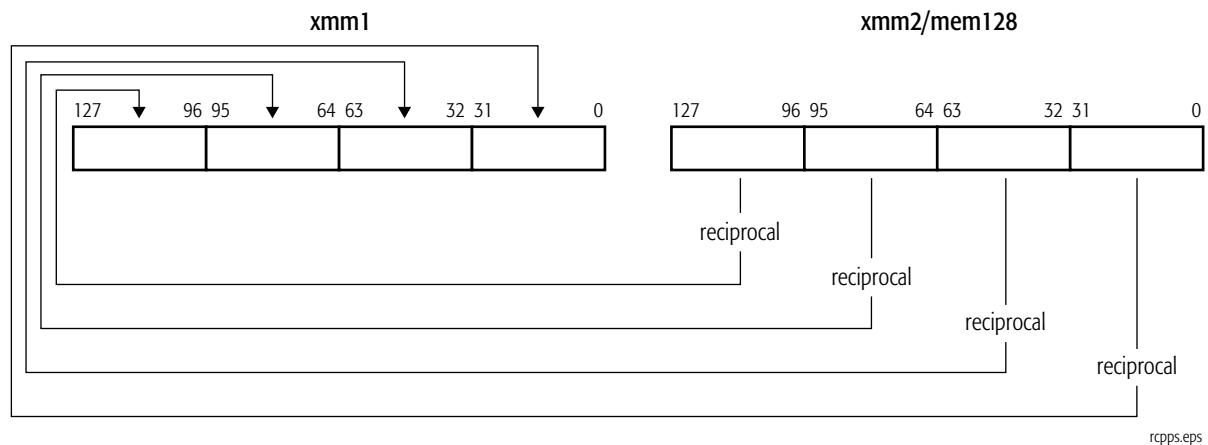
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

RCPPS**Reciprocal Packed Single-Precision Floating-Point**

The RCPPS instruction computes the approximate reciprocal of each of the four packed single-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the corresponding doubleword of another XMM register. The rounding control bits (RC) in the MXCSR register have no effect on the result.

The maximum relative error is less than or equal to 1.5×2^{-12} . A source value of 0.0 returns an infinity of the source value's sign. Denormal source operands are treated as signed 0.0. For negative source values other than 0.0, the QNaN floating-point indefinite value (“Indefinite Values” on page 161 of Volume 1) is returned. Results that underflow are changed to signed 0.0. For both SNaN and QNaN source operands, a QNaN is returned.

Mnemonic	Opcode	Description
RCPPS <i>xmm1, xmm2/mem128</i>	0F 53 /r	Computes reciprocals of packed single-precision floating-point values in an XMM register or 128-bit memory location and writes result in the destination XMM register.

**Related Instructions**

RCPS, RSQRTPS, RSQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

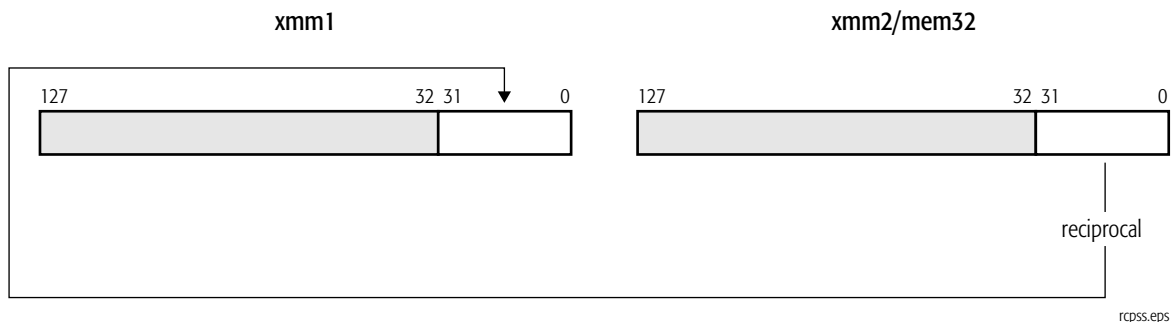
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

RCPSS**Reciprocal Scalar Single-Precision
Floating-Point**

The RCPSS instruction computes the approximate reciprocal of the low-order single-precision floating-point value in an XMM register or in a 32-bit memory location and writes the result in the low-order doubleword of another XMM register. The three high-order doublewords in the destination XMM register are not modified. The rounding control bits (RC) in the MXCSR register have no effect on the result.

The maximum relative error is less than or equal to 1.5×2^{-12} . A source value of 0.0 returns an infinity of the source value's sign. Denormal source operands are treated as signed 0.0. For negative source values other than 0.0, the QNaN floating-point indefinite value (“Indefinite Values” on page 161 of Volume 1) is returned. Results that underflow are changed to signed 0.0. For both SNaN and QNaN source operands, a QNaN is returned.

Mnemonic	Opcode	Description
RCPSS <i>xmm1, xmm2/mem32</i>	F3 0F 53 /r	Computes reciprocal of scalar single-precision floating-point value in an XMM register or 32-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

RCPPS, RSQRTPS, RSQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

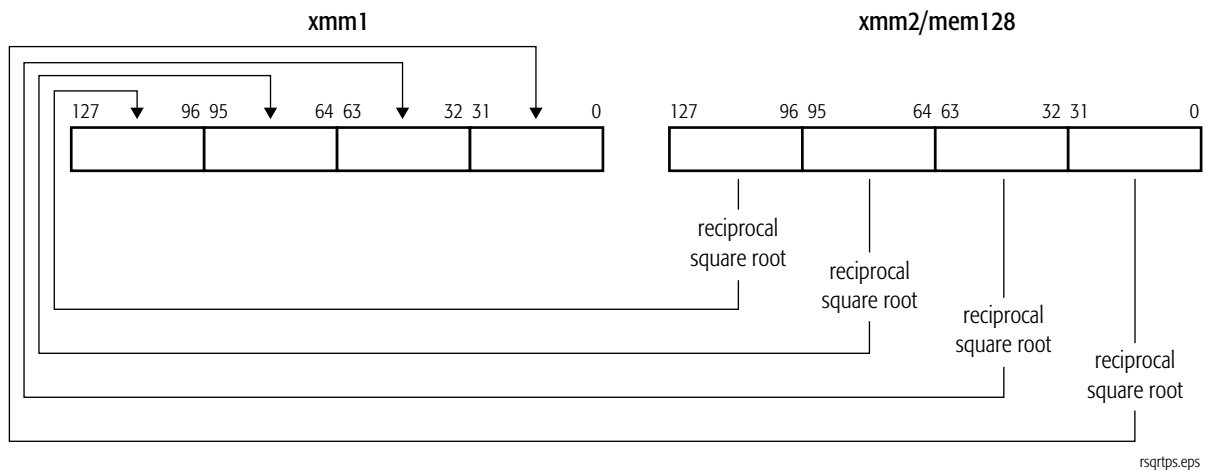
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

RSQRTPS**Reciprocal Square Root Packed Single-Precision Floating-Point**

The RSQRTPS instruction computes the approximate reciprocal of the square root of each of the four packed single-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the corresponding doubleword of another XMM register. The rounding control bits (RC) in the MXCSR register have no effect on the result.

The maximum relative error for the approximate reciprocal is less than or equal to 1.5×2^{-12} . A source value of 0.0 returns an infinity of the source value's sign. Denormal source operands are treated as signed 0.0. For negative source values other than 0.0, the QNaN floating-point indefinite value (“Indefinite Values” on page 161 of Volume 1) is returned. For both SNaN and QNaN source operands, a QNaN is returned.

Mnemonic	Opcode	Description
RSQRTPS <i>xmm1, xmm2/mem128</i>	0F 52 /r	Computes reciprocals of square roots of packed single-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

RSQRTSS, SQRTPD, SQRTPS, SQRTSD, SQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

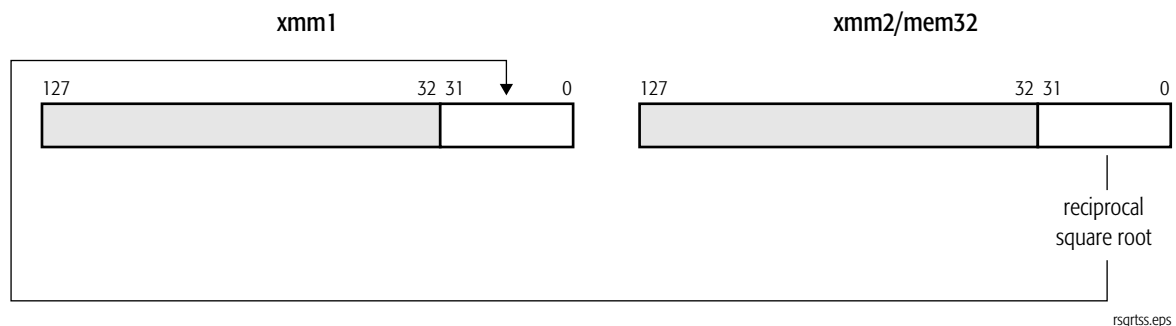
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

RSQRTSS**Reciprocal Square Root Scalar Single-Precision Floating-Point**

The RSQRTSS instruction computes the approximate reciprocal of the square root of the low-order single-precision floating-point value in an XMM register or in a 32-bit memory location and writes the result in the low-order doubleword of another XMM register. The three high-order doublewords in the destination XMM register are not modified. The rounding control bits (RC) in the MXCSR register have no effect on the result.

The maximum relative error for the approximate reciprocal is less than or equal to 1.5×2^{-12} . A source value of 0.0 returns an infinity of the source value's sign. Denormal source operands are treated as signed 0.0. For negative source values other than 0.0, the QNaN floating-point indefinite value (“Indefinite Values” on page 161 of Volume 1) is returned. For both SNaN and QNaN source operands, a QNaN is returned.

Mnemonic	Opcode	Description
RSQRTPS <i>xmm1, xmm2/mem32</i>	F3 0F 52/r	Computes reciprocal of square root of single-precision floating-point value in an XMM register or 32-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

RSQRTPS, SQRTPD, SQRTPS, SQRTSD, SQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

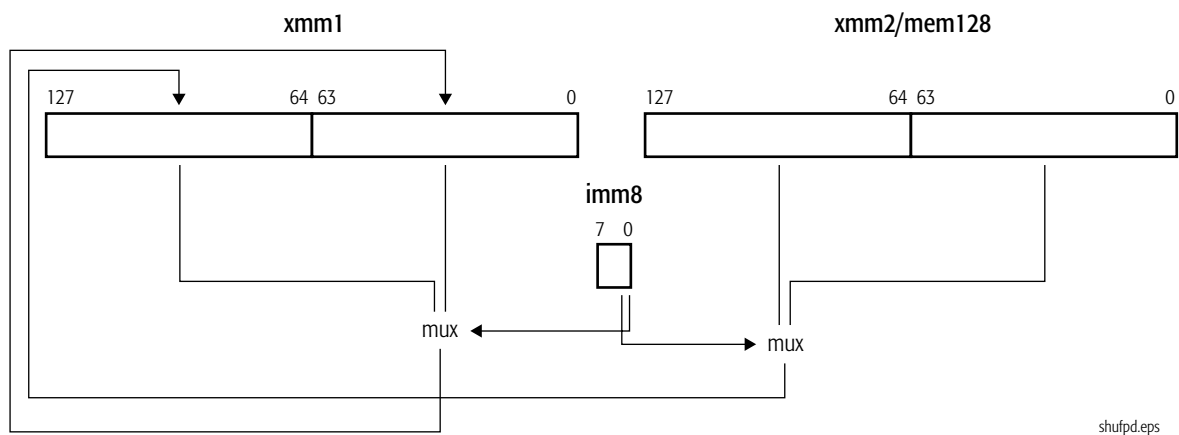
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

SHUFPD**Shuffle Packed Double-Precision Floating-Point**

The SHUFPD instruction moves either of the two packed double-precision floating-point values in the first source operand to the low-order quadword of the destination (first source) and moves either of the two packed double-precision floating-point values in the second source operand to the high-order quadword of the destination. In each case, the value of the destination quadword is determined by the least-significant two bits in the immediate-byte operand, as shown in Table 2-11. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
SHUFPD <i>xmm1, xmm2/mem128, imm8</i>	66 0F C6 /r <i>ib</i>	Shuffles packed double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and puts the result in the destination XMM register.



I

Table 2-11. Immediate-Byte Operand Encoding for SHUFPD

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source 1 Bits Moved	Source 2 Bits Moved
0	0	63:0	63:0	—
	1	63:0	127:64	—
1	0	127:64	—	63:0
	1	127:64	—	127:64

Related Instructions

SHUFPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

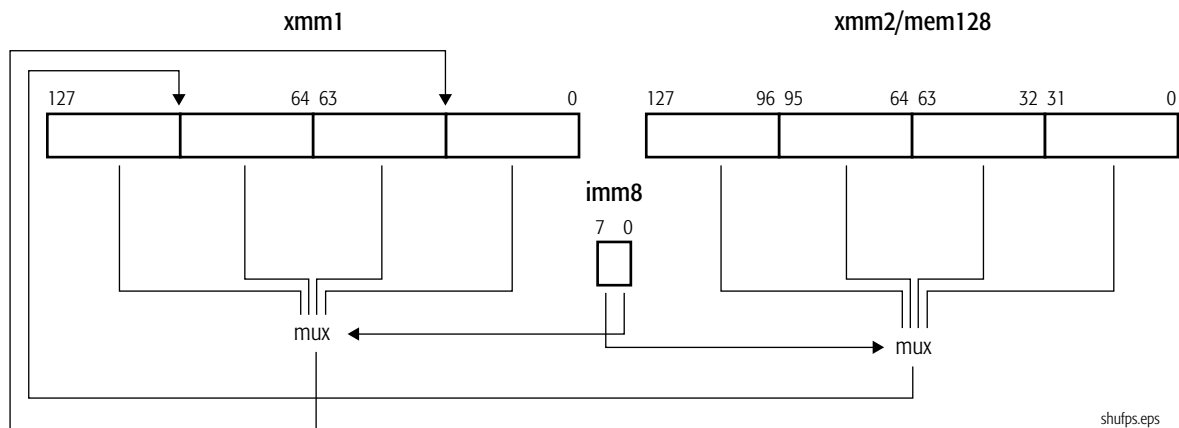
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

SHUFPS**Shuffle Packed Single-Precision Floating-Point**

The SHUFPS instruction moves two of the four packed single-precision floating-point values in the first source operand to the low-order quadword of the destination (first source) and moves two of the four packed single-precision floating-point values in the second source operand to the high-order quadword of the destination. In each case, the value of the destination doubleword is determined by a two-bit field in the immediate-byte operand, as shown in Table 2-12. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
SHUFPS <i>xmm1, xmm2/mem128, imm8</i>	0F C6/r <i>ib</i>	Shuffles packed single-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and puts the result in the destination XMM register.



shufps.eps

Table 2-12. Immediate-Byte Operand Encoding for SHUFPS

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source 1 Bits Moved	Source 2 Bits Moved
1:0	0	31:0	31:0	—
	1	31:0	63:32	—
	2	31:0	95:64	—
	3	31:0	127:96	—

Table 2-12. Immediate-Byte Operand Encoding for SHUFPS

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source 1 Bits Moved	Source 2 Bits Moved
3:2	0	63:32	31:0	—
	1	63:32	63:32	—
	2	63:32	95:64	—
	3	63:32	127:96	—
5:4	0	95:64	—	31:0
	1	95:64	—	63:32
	2	95:64	—	95:64
	3	95:64	—	127:96
7:6	0	127:96	—	31:0
	1	127:96	—	63:32
	2	127:96	—	95:64
	3	127:96	—	127:96

Related Instructions

SHUFPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

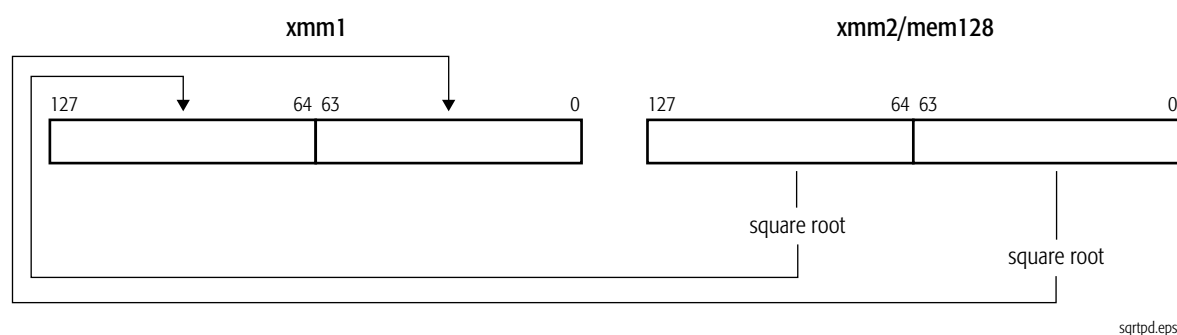
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

SQRTPD**Square Root Packed Double-Precision Floating-Point**

The SQRTPD instruction computes the square root of each of the two packed double-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the corresponding quadword of another XMM register.

Mnemonic	Opcode	Description
SQRTPD <i>xmm1, xmm2/mem128</i>	66 0F 51 /r	Computes square roots of packed double-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

RSQRTPS, RSQRTSS, SQRTPS, SQRTSD, SQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0				1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.																		

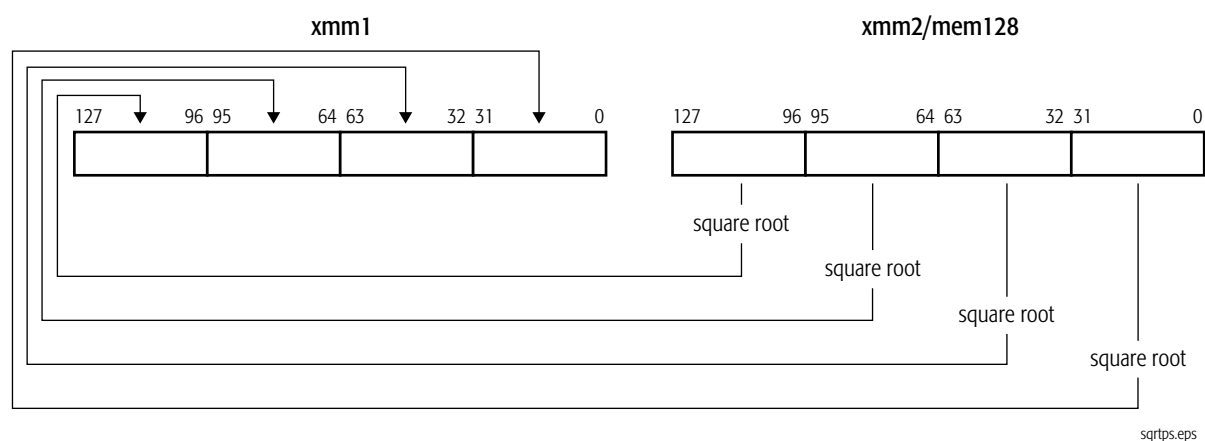
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or is less than 0 (not including -0).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SQRTPS**Square Root Packed Single-Precision Floating-Point**

The SQRTPS instruction computes the square root of each of the four packed single-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the corresponding doubleword of another XMM register.

Mnemonic	Opcode	Description
SQRTPS <i>xmm1, xmm2/mem128</i>	0F 51 /r	Computes square roots of packed single-precision floating-point values in an XMM register or 128-bit memory location and writes the result in the destination XMM register.

**Related Instructions**

RSQRTPS, RSQRTSS, SQRTPD, SQRTSD, SQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0				1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

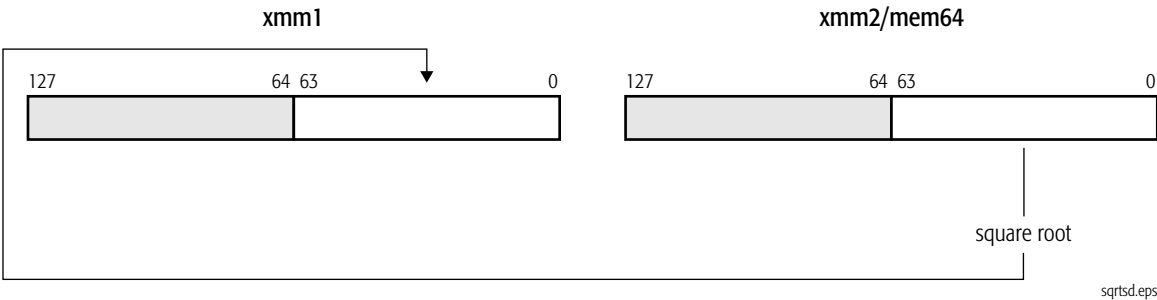
Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or is less than 0 (not including -0).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SQRTSD

Square Root Scalar Double-Precision Floating-Point

The SQRTSD instruction computes the square root of the low-order double-precision floating-point value in an XMM register or in a 64-bit memory location and writes the result in the low-order quadword of another XMM register. The high-order quadword of the destination XMM register is not modified.

Mnemonic	Opcode	Description
SQRTSD <i>xmm1, xmm2/mem64</i>	F2 0F 51 /r	Computes square root of double-precision floating-point value in an XMM register or 64-bit memory location and writes the result in the destination XMM register.



Related Instructions

RSQRTPS, RSQRTSS, SQRTPD, SQRTPS, SQRTSS

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0				1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.																	

Exceptions

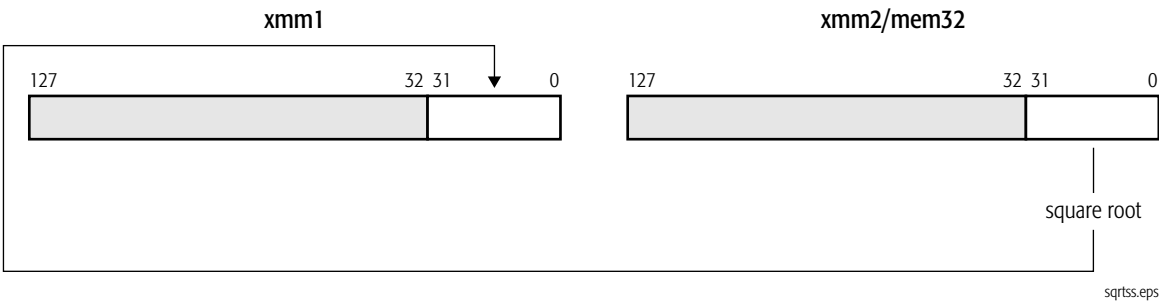
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or is less than 0 (not including -0).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SQRTSS

Square Root Scalar Single-Precision Floating-Point

The SQRTSS instruction computes the square root of the low-order single-precision floating-point value in an XMM register or 32-bit memory location and writes the result in the low-order doubleword of another XMM register. The three high-order doublewords of the destination XMM register are not modified.

Mnemonic	Opcode	Description
SQRTSS <i>xmm1, xmm2/mem32</i>	F3 0F 51 /r	Computes square root of single-precision floating-point value in an XMM register or 32-bit memory location and writes the result in the destination XMM register.



Related Instructions

RSQRTPS, RSQRTSS, SQRTPD, SQRTPS, SQRTSD

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0				1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.																	

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or is less than 0 (not including -0).
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

STMXCSR**Store MXCSR Control/Status Register**

The STMXCSR instruction saves the contents of the MXCSR register in a 32-bit location in memory. The MXCSR register is described in Section 4.3, “Registers”, on page 143 of Volume 1.

Mnemonic	Opcode	Description
STMXCSR <i>mem32</i>	0F AE /3	Stores contents of MXCSR in 32-bit memory location.

Related Instructions

LDMXCSR

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

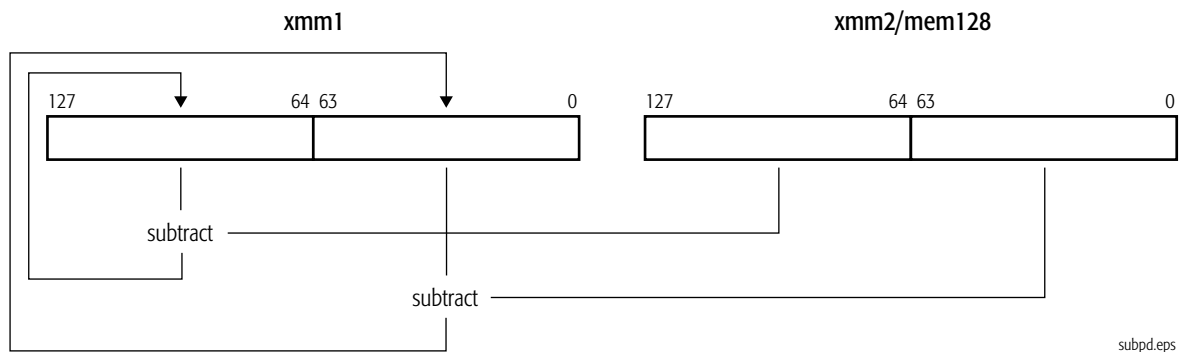
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

SUBPD**Subtract Packed Double-Precision Floating-Point**

The SUBPD instruction subtracts each packed double-precision floating-point value in the second source operand from the corresponding packed double-precision floating-point value in the first source operand and writes the result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
SUBPD <i>xmm1, xmm2/mem128</i>	66 0F 5C /r	Subtracts packed double-precision floating-point values in an XMM register or 128-bit memory location from packed double-precision floating-point values in another XMM register and writes the result in the destination XMM register.

**Related Instructions**

SUBPS, SUBSD, SUBSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

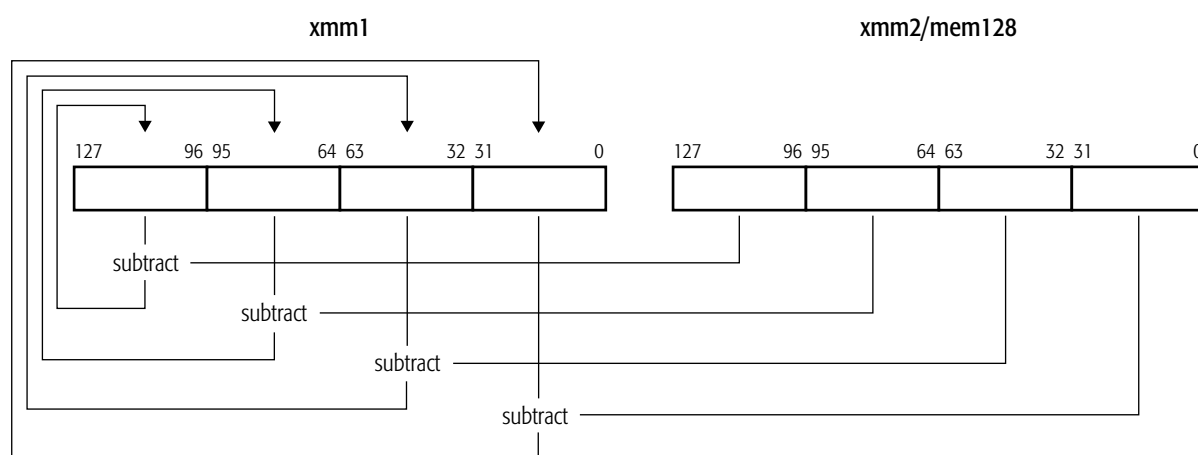
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or both source operands are infinities of the same sign.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SUBPS**Subtract Packed Single-Precision Floating-Point**

The SUBPS instruction subtracts each packed single-precision floating-point value in the second source operand from the corresponding packed single-precision floating-point value in the first source operand and writes the result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
SUBPS <i>xmm1, xmm2/mem128</i>	0F 5C /r	Subtracts packed single-precision floating-point values in an XMM register or 128-bit memory location from packed single-precision floating-point values in another XMM register and writes the result in the destination XMM register.

**Related Instructions**

SUBPD, SUBSD, SUBSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

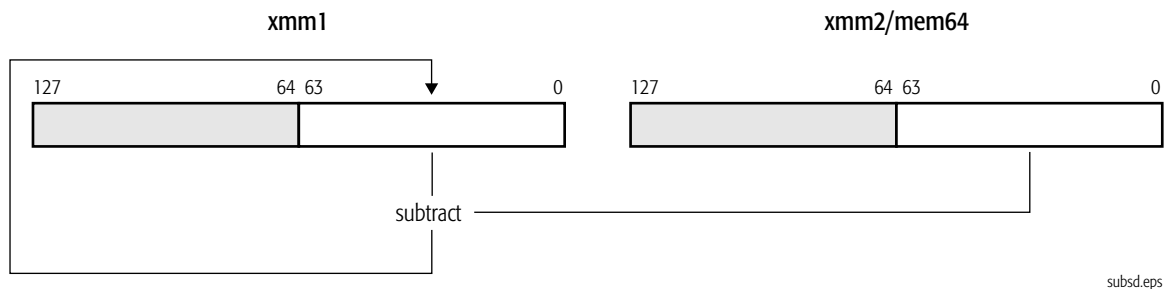
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or both source operands are infinities of the same sign.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SUBSD**Subtract Scalar Double-Precision Floating-Point**

The SUBSD instruction subtracts the double-precision floating-point value in the low-order quadword of the second source operand from the double-precision floating-point value in the low-order quadword of the first source operand and writes the result in the low-order quadword of the destination (first source). The high-order quadword of the destination is not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 64-bit memory location.

Mnemonic	Opcode	Description
SUBSD <i>xmm1, xmm2/mem64</i>	F2 0F 5C/ <i>r</i>	Subtracts low-order double-precision floating-point value in an XMM register or in a 64-bit memory location from low-order double-precision floating-point value in another XMM register and writes the result in the destination XMM register.

**Related Instructions**

SUBPD, SUBPS, SUBSS

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

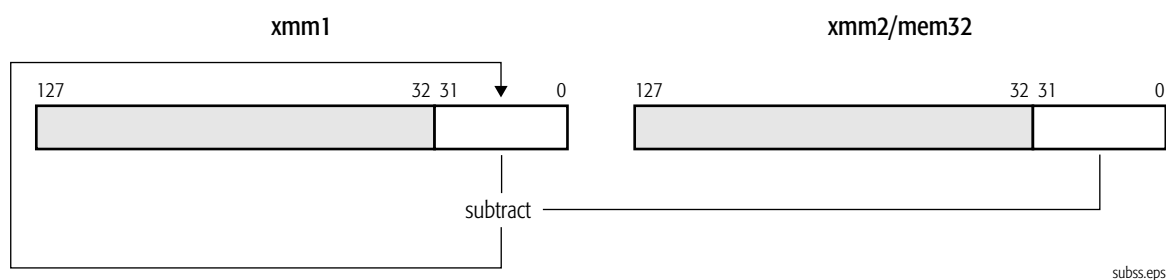
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or both source operands are infinities of the same sign.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

SUBSS**Subtract Scalar Single-Precision Floating-Point**

The SUBSS instruction subtracts the single-precision floating-point value in the low-order doubleword of the second source operand from the single-precision floating-point value in the low-order doubleword of the first source operand and writes the result in the low-order doubleword of the destination (first source). The three high-order doublewords of the destination are not modified. The first source/destination operand is an XMM register. The second source operand is another XMM register or 32-bit memory location.

Mnemonic	Opcode	Description
SUBSS <i>xmm1</i> , <i>xmm2/mem32</i>	F3 0F 5C/ <i>r</i>	Subtracts low-order single-precision floating-point value in an XMM register or in a 32-bit memory location from low-order single-precision floating-point value in another XMM register and writes the result in the destination XMM register.

**Related Instructions**

SUBPD, SUBPS, SUBSD

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0	1/0	1/0		1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank. Reserved fields are shaded.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

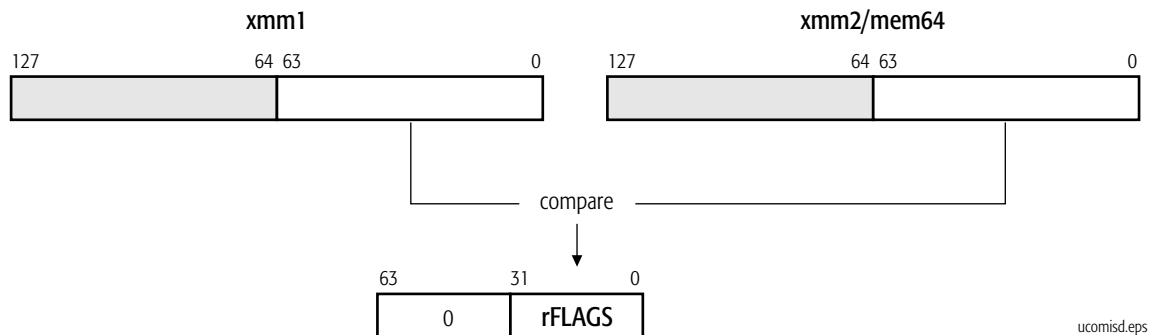
Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or both source operands are infinities of the same sign.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the format of the destination operand (either tiny, or if masked, tiny and inexact).
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

UCOMISD**Unordered Compare Scalar
Double-Precision Floating-Point**

The UCOMISD instruction performs an unordered compare of the double-precision floating-point value in the low-order 64 bits of an XMM register with the double-precision floating-point value in the low-order 64 bits of another XMM register or a 64-bit memory location and sets the ZF, PF, and CF bits in the rFLAGS register to reflect the result of the compare. The result is unordered if one or both of the operand values is a NaN. The OF, AF, and SF bits in rFLAGS are set to zero.

If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

Mnemonic	Opcode	Description
UCOMISD <i>xmm1, xmm2/mem64</i>	66 0F 2E/r	Compares scalar double-precision floating-point values in an XMM register and an XMM register or 64-bit memory location. Sets rFLAGS.



ucomisd.eps

Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

Related Instructions

CMPPD, CMPPS, CMPSD, CMPSS, COMISD, COMISS, UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	1/0	0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank.
 If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

MXCSR Flags Affected

		FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																	1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

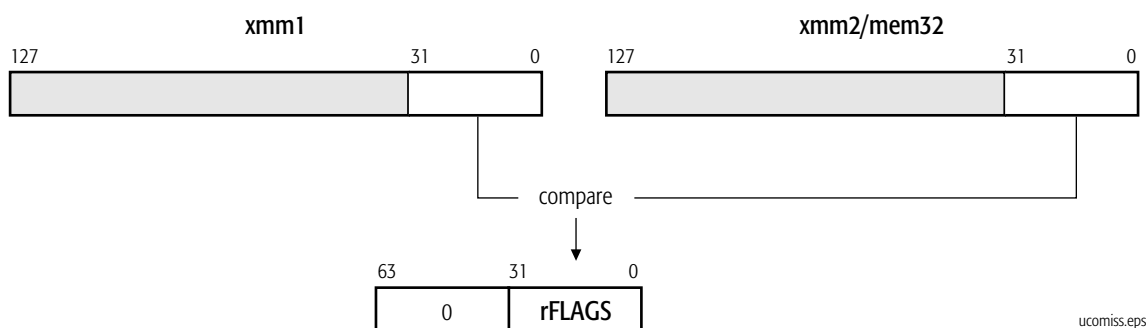
Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or unsupported format.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

UCOMISS**Unordered Compare Scalar
Single-Precision Floating-Point**

The UCOMISS instruction performs an unordered compare of the single-precision floating-point value in the low-order 32 bits of an XMM register with the single-precision floating-point value in the low-order 32 bits of another XMM register or a 32-bit memory location and sets the ZF, PF, and CF bits in the rFLAGS register to reflect the result. The result is unordered if one or both of the operand values is a NaN. The OF, AF, and SF bits in rFLAGS are set to zero.

If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

Mnemonic	Opcode	Description
UCOMISS <i>xmm1, xmm2/mem32</i>	0F 2E/r	Compares scalar single-precision floating-point values in an XMM register and an XMM register or 32-bit memory location. Sets rFLAGS.



Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

Related Instructions

CMPPD, CMPPS, CMPSD, CMPSS, COMISD, COMISS, UCOMISD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	1/0	0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:

Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to 1 or cleared to 0 is 1/0. Unaffected flags are blank.
If the instruction causes an unmasked SIMD floating-point exception (#XM), the rFLAGS bits are not updated.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
																1/0	1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:

A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

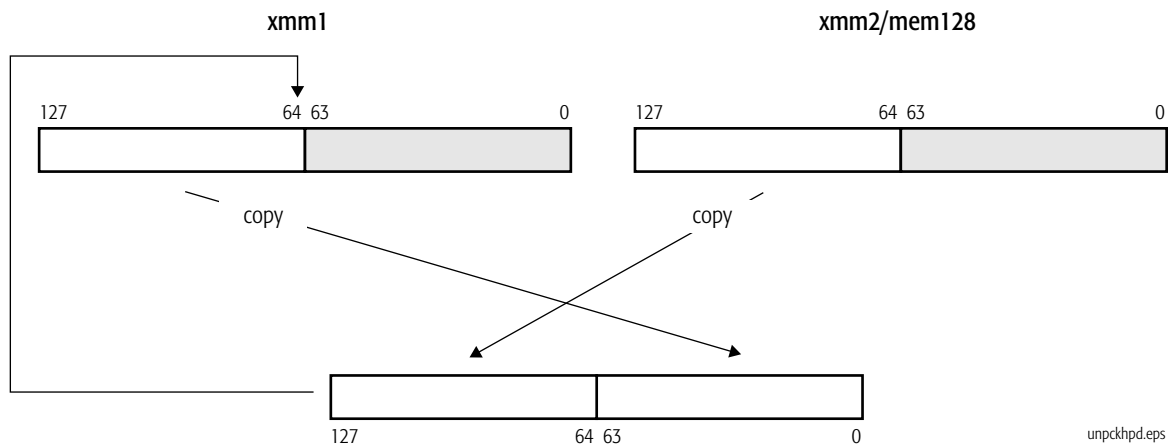
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	A source operand is an SNaN value or unsupported format.
Denormalized-operand exception (DE)	X	X	X	A source operand is a denormal value.

UNPCKHPD**Unpack High Double-Precision Floating-Point**

The UNPCKHPD instruction unpacks the high-order double-precision floating-point values in the first and second source operands and packs them into quadwords in the destination (first source). The value from the first source operand is packed into the low-order quadword of the destination, and the value from the second source operand is packed into the high-order quadword of the destination. The low-order quadwords of the source operands are ignored. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
UNPCKHPD <i>xmm1, xmm2/mem128</i>	66 0F 15 /r	Unpacks high-order double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and packs them into the destination XMM register.

**Related Instructions**

UNPCKHPS, UNPCKLPD, UNPCKLPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

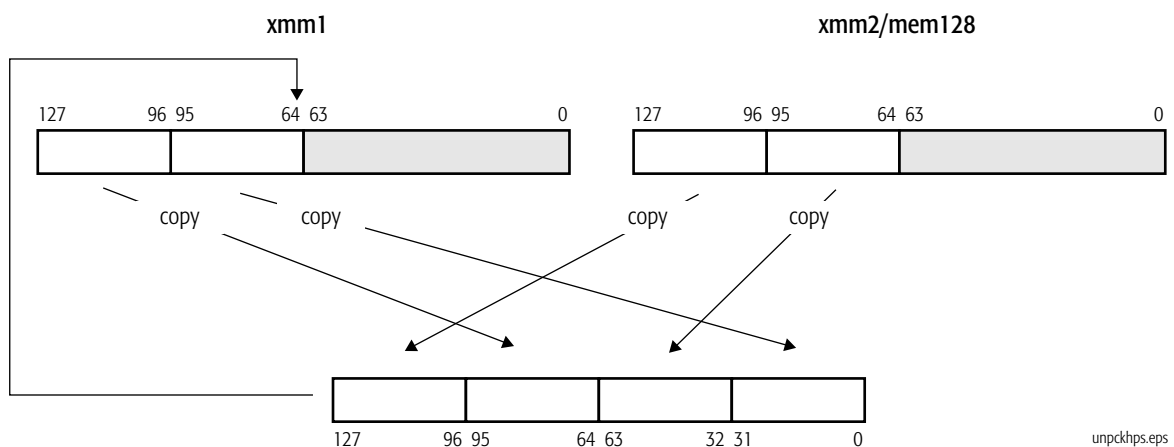
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

UNPCKHPS**Unpack High Single-Precision Floating-Point**

The UNPCKHPS instruction unpacks the high-order single-precision floating-point values in the first and second source operands and packs them into interleaved doublewords in the destination (first source). The low-order quadwords of the source operands are ignored. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
UNPCKHPS <i>xmm1, xmm2/mem128</i>	0F 15/r	Unpacks high-order single-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and packs them into the destination XMM register.



unpckhps.eps

Related Instructions

UNPCKHPD, UNPCKLPD, UNPCKLPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

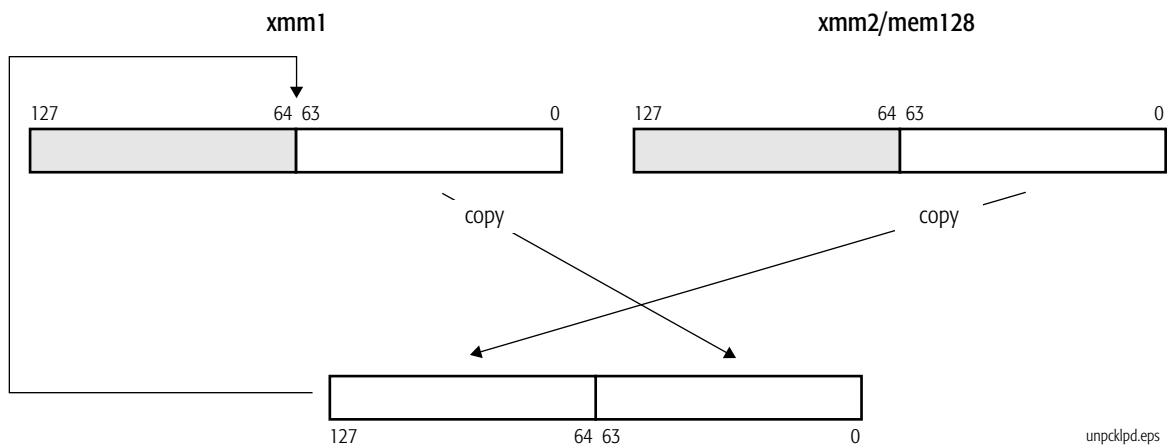
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

UNPCKLPD**Unpack Low Double-Precision Floating-Point**

The UNPCKLPD instruction unpacks the low-order double-precision floating-point values in the first and second source operands and packs them into the destination (first source). The value from the first source operand is packed into the low-order quadword of the destination, and the value from the second source operand is packed into the high-order quadword of the destination. The high-order quadwords of the source operands are ignored. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
UNPCKLPD <i>xmm1, xmm2/mem128</i>	66 0F 14/r	Unpacks low-order double-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and packs them into the destination XMM register.



unpcklpd.eps

Related Instructions

UNPCKHPD, UNPCKHPS, UNPCKLPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

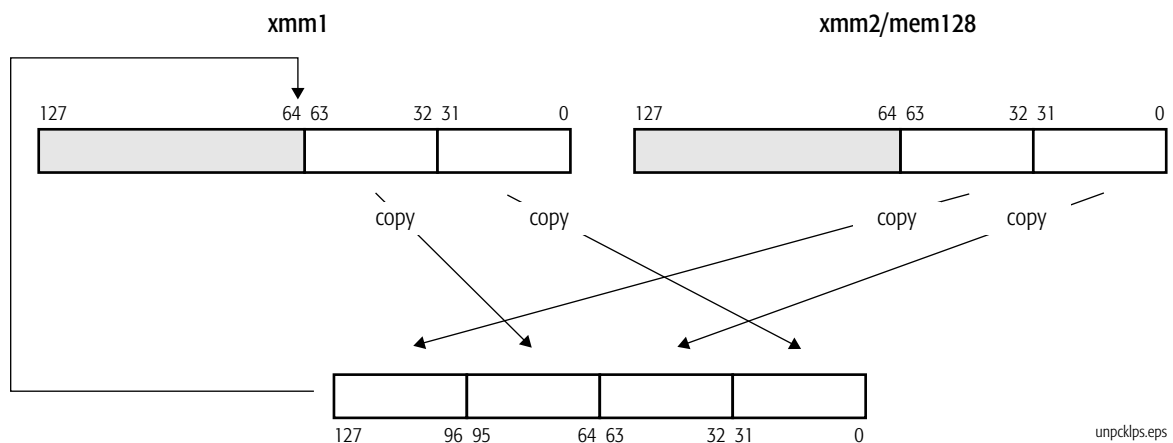
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

UNPCKLPS**Unpack Low Single-Precision Floating-Point**

The UNPCKLPS instruction unpacks the low-order single-precision floating-point values in the first and second source operands and packs them into interleaved doublewords in the destination (first source). The high-order quadwords of the source operands are ignored. The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location

Mnemonic	Opcode	Description
UNPCKLPS <i>xmm1, xmm2/mem128</i>	OF 14/ <i>r</i>	Unpacks low-order single-precision floating-point values in an XMM register and another XMM register or 128-bit memory location and packs them into the destination XMM register.

**Related Instructions**

UNPCKHPD, UNPCKHPS, UNPCKLPD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

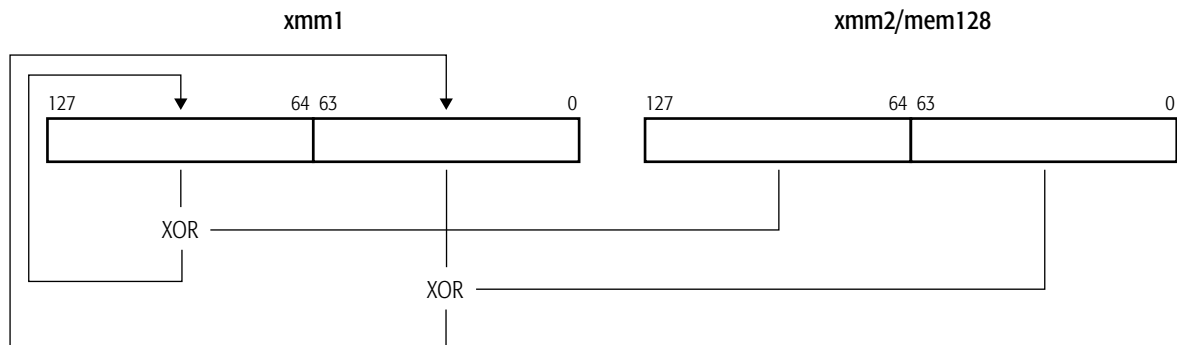
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

XORPD**Logical Bitwise Exclusive OR
Packed Double-Precision Floating-Point**

The XORPD instruction performs a bitwise logical Exclusive OR of the two packed double-precision floating-point values in the first source operand and the corresponding two packed double-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
XORPD <i>xmm1, xmm2/mem128</i>	66 0F 57 /r	Performs bitwise logical XOR of two packed double-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



xorpd.eps

Related Instructions

ANDNPD, ANDNPS, ANDPD, ANDPS, ORPD, ORPS, XORPS

rFLAGS Affected

None.

MXCSR Flags Affected

None.

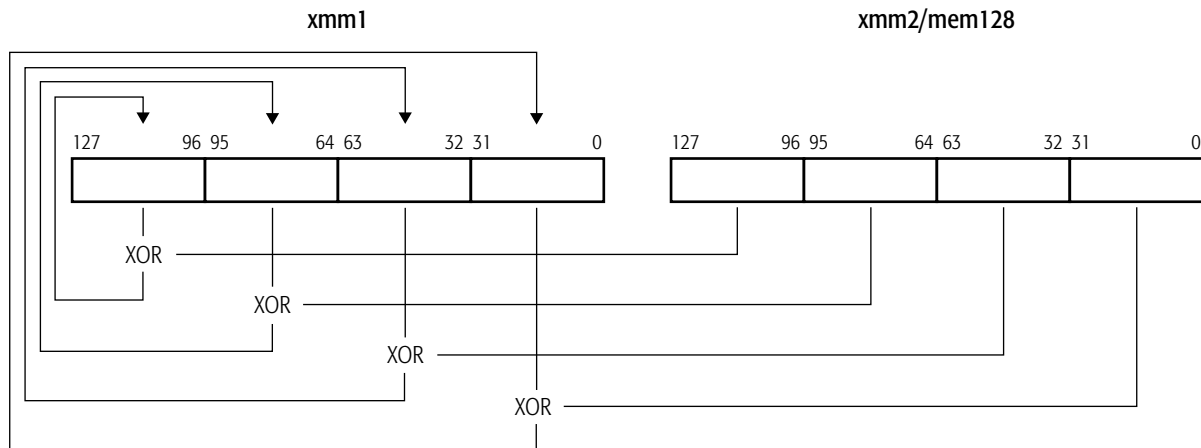
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

XORPS**Logical Bitwise Exclusive OR
Packed Single-Precision Floating-Point**

The XORPS instruction performs a bitwise Exclusive OR of the four packed single-precision floating-point values in the first source operand and the corresponding four packed single-precision floating-point values in the second source operand and writes the result in the destination (first source). The first source/destination operand is an XMM register. The second source operand is another XMM register or 128-bit memory location.

Mnemonic	Opcode	Description
XORPS <i>xmm1, xmm2/mem128</i>	0F 57 /r	Performs bitwise logical XOR of four packed single-precision floating-point values in an XMM register and in another XMM register or 128-bit memory location and writes the result in the destination XMM register.



xorps.eps

Related Instructions

ANDNPD, ANDNPS, ANDPD, ANDPS, ORPD, ORPS, XORPD

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

2.8 64-Bit Media Instruction Reference

This section describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by the 64-bit media instructions. These instructions operate on data located in the 64-bit MMX™ registers. Most of the instructions operate in parallel on sets of packed elements called *vectors*, although some operate on scalars. The instructions define both integer and floating-point operations, and include the legacy MMX instructions, the 3DNow!™ instructions, and the AMD extensions to the MMX and 3DNow! instruction sets.

Each instruction that performs a vector (packed) operation is illustrated with a diagram. Figure 2-16 shows the conventions used in these diagrams. The particular diagram shows the PSSLW (packed shift left logical words) instruction.

Arrowheads going to a source operand indicate the writing of the result. In this case, the result is written to the first source operand, which is also the destination operand.

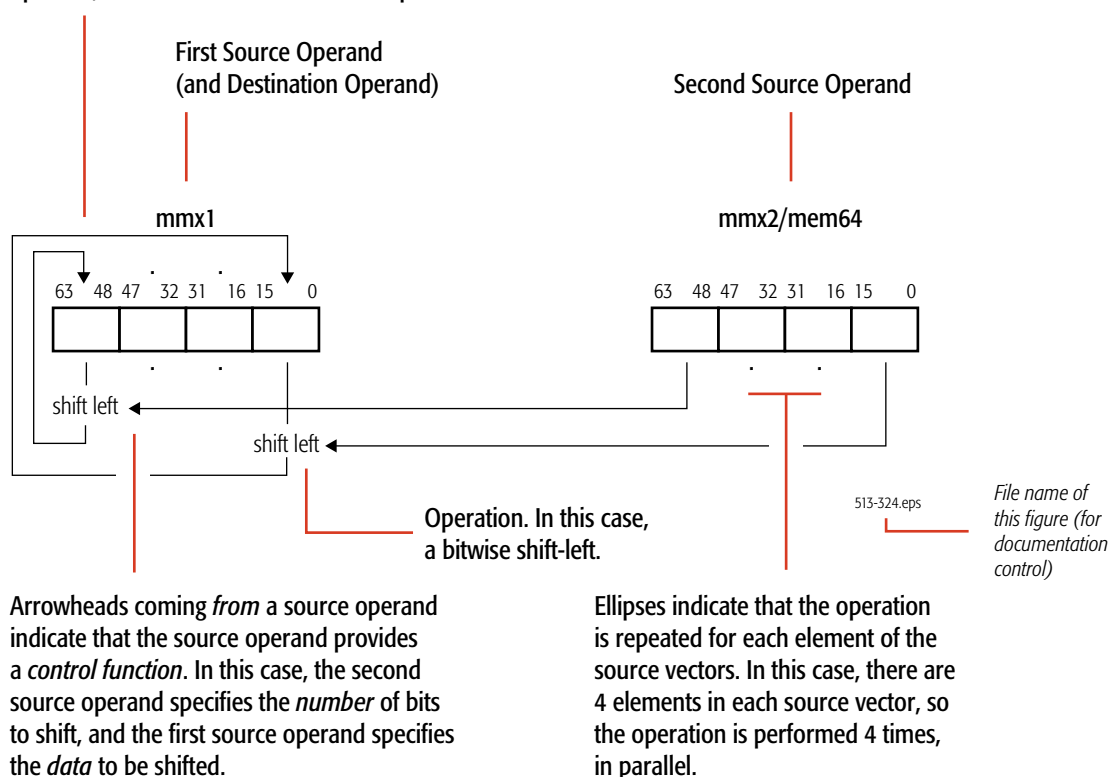


Figure 2-16. Diagram Conventions for 64-Bit Media Instructions

Gray areas in diagrams indicate unmodified operand bits.

Like the 128-bit media instructions, many of the 64-bit instructions independently and simultaneously perform a single operation on multiple elements of a vector and are thus classified as *single-instruction, multiple-data* (SIMD) instructions. A few 64-bit media instructions convert operands in MMX registers to operands in GPR, XMM, or x87 registers (or vice versa), or save or restore MMX state, or reset x87 state.

Hardware support for a specific 64-bit media instruction depends on the presence of at least one of the following CPUID functions:

- MMX Instructions, indicated by bit 23 of CPUID standard function 1 and extended function 8000_0001h.
- AMD Extensions to MMX Instructions, indicated by bit 22 of CPUID extended function 8000_0001h.
- SSE, indicated by bit 25 of CPUID extended function 8000_0001h.
- AMD 3DNow! Instructions, indicated by bit 31 of CPUID extended function 8000_0001h.
- AMD Extensions to 3DNow! Instructions, indicated by bit 30 of CPUID extended function 8000_0001h.
- FXSAVE and FXRSTOR, indicated by bit 24 of CPUID standard function 1 and extended function 8000_0001h.

The 64-bit media instructions can be used in legacy mode or long mode. Their use in long mode is available if the following CPUID function is set:

- Long Mode, indicated by bit 29 of CPUID extended function 8000_0001h.

Compilation of 64-bit media programs for execution in 64-bit mode offers four primary advantages: access to the eight extended, 64-bit general-purpose registers (for a register set consisting of GPR0–GPR15), access to the eight extended XMM registers (for a register set consisting of XMM0–XMM15), access to the 64-bit virtual address space, and access to the RIP-relative addressing mode.

For further information, see:

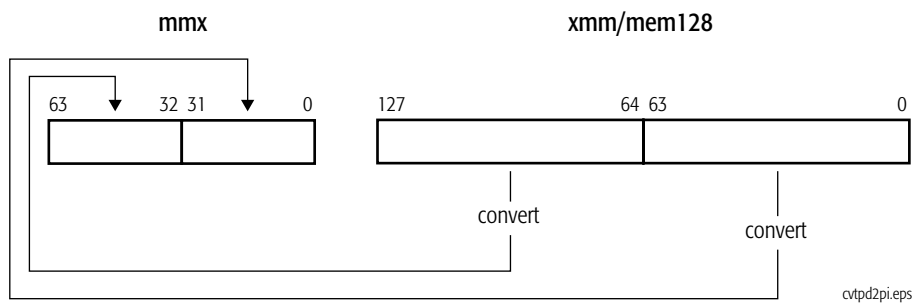
- Volume 1: Section 5, “64-Bit Media Programming”, on page 233.

- Volume 3: Section 2.3, “Summary of Registers and Data Types”, on page 28.
- Volume 3: Section 2.5, “Notation”, on page 41.
- Volume 3: Section 1.2, “Instruction Prefixes”, on page 3.

CVTPD2PI**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers**

The CVTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPD2PI <i>mmx, xmm2/mem128</i>	66 0F 2D /r	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integers values in the destination MMX register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPI2PD, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

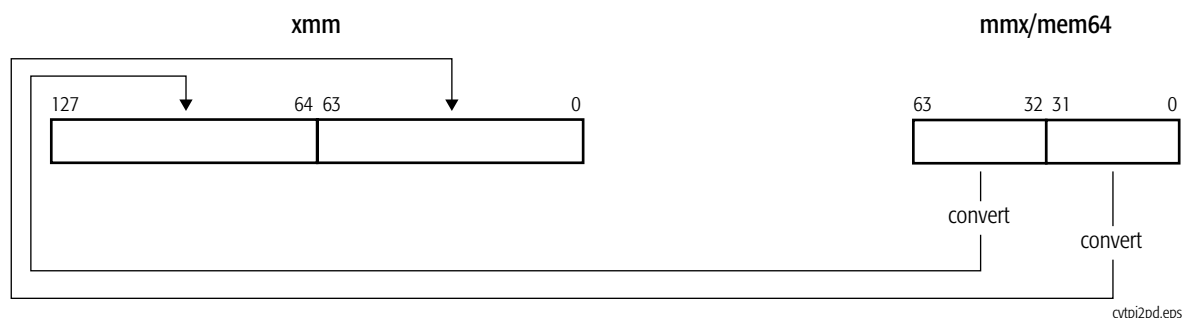
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPI2PD**Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point**

The CVTPI2PD instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two double-precision floating-point values and writes the converted values in an XMM register.

Mnemonic	Opcode	Description
CVTPI2PD <i>xmm, mmx/mem64</i>	66 0F 2A/r	Converts two packed doubleword integer values in an MMX register or 64-bit memory location to two packed double-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTPD2PI, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

None.

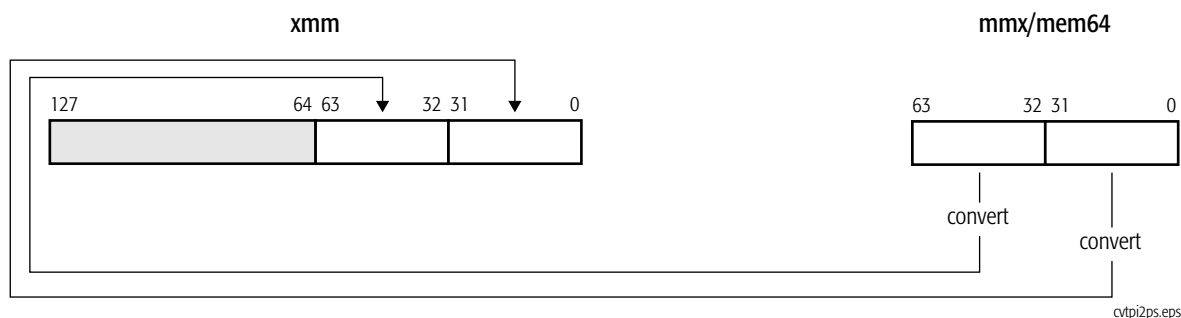
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPI2PS**Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point**

The CVTPI2PS instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two single-precision floating-point values and writes the converted values in the low-order 64 bits of an XMM register. The high-order 64 bits of the XMM register are not modified.

Mnemonic	Opcode	Description
CVTPI2PS <i>xmm, mmx/mem64</i>	0F 2A/ <i>r</i>	Converts packed doubleword integer values in an MMX register or 64-bit memory location to single-precision floating-point values in the destination XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

	FZ	RC			PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

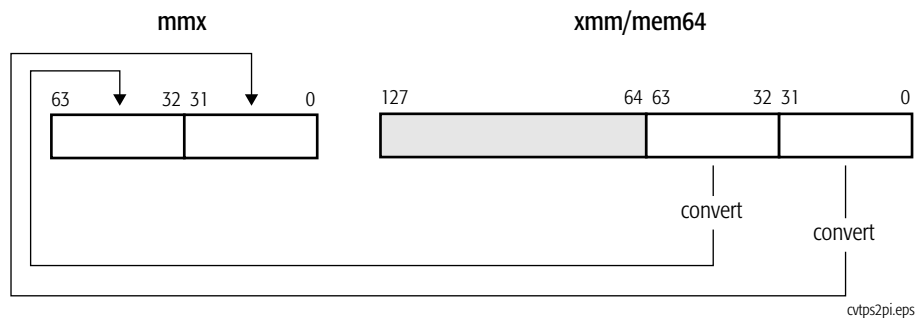
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTPS2PI**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers**

The CVTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integers and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPS2PI <i>mmx, xmm/mem64</i>	0F 2D /r	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to packed doubleword integers in the destination MMX register.



If the result of the conversion is an inexact value, the value is rounded as specified by the rounding control bits (RC) in the MXCSR register. If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word and the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTPS2PI, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

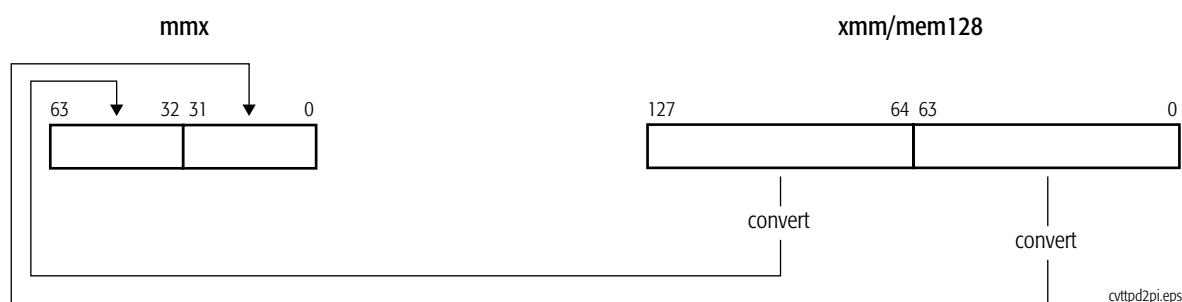
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPD2PI**Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPD2PI instruction converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTPD2PI <i>mmx, xmm/mem128</i>	66 0F 2C/ <i>r</i>	Converts packed double-precision floating-point values in an XMM register or 128-bit memory location to packed doubleword integer values in the destination MMX register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PD, CVTPD2DQ, CVTPD2PI, CVTPD2SI, CVTSD2SI, CVTSI2SD, CVTTPD2DQ, CVTTSD2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ		RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
													1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Note: A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.																		

Exceptions

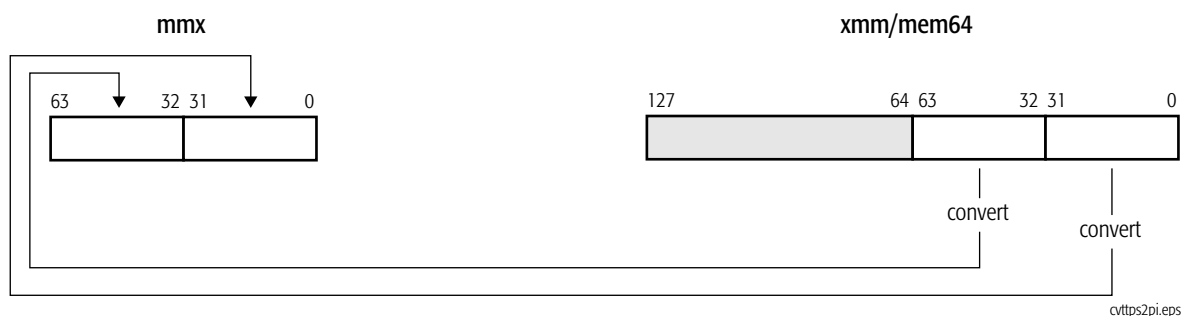
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

CVTTPS2PI**Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated**

The CVTTPS2PI instruction converts two packed single-precision floating-point values in the low-order 64 bits of an XMM register or a 64-bit memory location to two packed 32-bit signed integer values and writes the converted values in an MMX register.

Mnemonic	Opcode	Description
CVTTPS2PI <i>mmx xmm/mem64</i>	0F 2C/r	Converts packed single-precision floating-point values in an XMM register or 64-bit memory location to doubleword integer values in the destination MMX register. Inexact results are truncated.



If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). If the floating-point value is a NaN, infinity, or if the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

CVTDQ2PS, CVTPI2PS, CVTPS2DQ, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2DQ, CVTTSS2SI

rFLAGS Affected

None.

MXCSR Flags Affected

		FZ	RC		PM	UM	OM	ZM	DM	IM		PE	UE	OE	ZE	DE	IE
												1/0					1/0
31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note:
A flag that can be set to one or zero is 1/0. Unaffected flags are blank. Shaded fields are reserved.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is cleared to 0, and there is an unmasked SIMD floating-point exception. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
SIMD Floating-Point Exception, #XM	X	X	X	The operating-system unmasked-exception support bit (OSXMMEXCPT) of CR4 is set to 1, and there is an unmasked SIMD floating-point exception. See <i>SIMD Floating-Point Exceptions</i> , below, for details.
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN, QNaN, or infinity, or there is an overflow.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

EMMS

Exit Multimedia State

The EMMS instruction clears the MMX state by setting the state of the x87 stack registers to *empty* (tag-bit encoding of all 1s for all MMX registers) indicating that the contents of the registers are available for a new procedure, such as an x87 floating-point procedure. This setting of the tag bits is referred to as “clearing the MMX state”.

Because the MMX registers and tag word are shared with the x87 floating-point instructions, software should execute an EMMS instruction to clear the MMX state before executing code that includes x87 floating-point instructions.

The functions of the EMMS and FEMMS instructions are identical.

For details about the setting of x87 tag bits, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
EMMS	0F 77	Clears the MMX state.

Related Instructions

FEMMS (a 3DNow! instruction)

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

FEMMS

Fast Exit Multimedia State

The FEMMS instruction clears the MMX state by setting the state of the x87 stack registers to *empty* (tag-bit encoding of all 1s for all MMX registers) indicating that the contents of the registers are available for a new procedure, such as an x87 floating-point procedure. This setting of the tag bits is referred to as “clearing the MMX state”.

Because the MMX registers and tag word are shared with the x87 floating-point instructions, software should execute an EMMS or FEMMS instruction to clear the MMX state before executing code that includes x87 floating-point instructions.

FEMMS is a 3DNow! instruction. The functions of the FEMMS and EMMS instructions are identical. The FEMMS instruction is supported for backward-compatibility with certain AMD processors. Software that must be both compatible with both AMD and non-AMD processors should use the EMMS instruction.

For details about the setting of x87 tag bits, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FEMMS	0F 0E	Clears MMX state.

Related Instructions

EMMS

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

FNSAVE**Save No-Wait x87 and MMX State**

The FNSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and reinitializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Unlike FSAVE, FNSAVE does not wait for pending unmasked x87 floating-point exceptions to occur. This no-wait instruction should be used when processor interrupts are disabled and x87 instructions might generate an interrupt that would create an endless wait.

For details about the memory image saved by FNSAVE, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FNSAVE <i>mem94/108env</i>	DD /6	Store a copy of the floating-pointing state to <i>mem94/108env</i> without checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples**1. Save environment and stack data:**

```
FNSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FSAVE, FRSTOR, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Description
C0	0
C1	0
C2	0
C3	0

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

FRSTOR**Restore x87 and MMX State**

The FRSTOR instruction restores complete x87 state from a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit. The MMX state is also restored by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are loaded from memory.

Unlike the FXRSTOR instruction, FRSTOR checks for pending unmasked x87 floating-point exceptions in the restored image. However, if such a pending unmasked exception is found, the exception does not occur immediately after execution of an FRSTOR instruction. Instead, the processor asserts the FERR# (floating-point error) output signal. The normal rules for x87 exceptions then apply, just as if the exception was created by an x87 instruction. If CRO.NE = 1, the processor vectors to the x87 floating-point exception (#MF) handler at the next non-wait x87 instruction. Otherwise, the processor examines the state of the IGNNE# (ignore numeric error) input signal. If IGNNE# is asserted, the processor executes the next x87 instruction. If IGNNE# is negated, the processor freezes at that instruction boundary until an external interrupt occurs that redirects control to a handler that clears the exception.

For details about the memory image restored by FRSTOR, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FRSTOR <i>mem94/108env</i>	DD /4	Load a copy of the floating-pointing state from <i>mem94/108env</i> .

Examples**1. Save environment and stack data:**

```

FSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FSAVE, FNSAVE, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Description
C0	Loaded from memory.
C1	Loaded from memory.
C2	Loaded from memory.
C3	Loaded from memory.

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

FSAVE**Save x87 and MMX State**

The FSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and initializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Assemblers issue FSAVE as an FWAIT instruction (9Bh) followed by an FNSAVE instruction. Thus, FSAVE (but not FNSAVE) reports pending unmasked x87 floating-point exceptions before saving the state.

For details about the memory image saved by FSAVE, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FSAVE <i>mem94/108env</i>	9B DD /6	Store a copy of the floating-pointing state to <i>mem94/108env</i> after checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples**1. Save environment and stack data:**

```

FSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr     ;Restore the old operating environment.
```

Related Instructions

FNSAVE, FRSTOR, FXSAVE, FXRSTOR

x87 Condition Code

x87 Condition Code	Value	Description
C0	0	
C1	0	
C2	0	
C3	0	
<i>U indicates 'undefined.'</i>		

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

FXRSTOR**Restore XMM, MMX, and x87 State**

The FXRSTOR instruction restores the XMM, MMX, and x87 state. The data loaded from memory is the state information previously saved using the FXSAVE instruction. Restoring data with FXRSTOR that had been previously saved with an FSAVE (rather than FXSAVE) instruction results in an incorrect restoration.

Mnemonic	Opcode	Description
FXRSTOR <i>mem512env</i>	OF AE /1	Restores XMM, MMX, and x87 state.

Unlike the FRSTOR instruction, FXRSTOR does not cause pending unmasked x87 floating-point exceptions in the restored image to be recognized when internal control of x87 exceptions is enabled (CR0.NE=1). When loading a new environment, use an FWAIT instruction after the FXRSTOR instruction to check for and handle any pending unmasked x87 exceptions.

If the restored MXCSR register contains a set bit in an exception status flag, and the corresponding exception mask bit is cleared (indicating an unmasked exception), loading the MXCSR register from memory does not cause a SIMD floating-point exception (#XM).

FXRSTOR executes faster than FRSTOR because it does not restore the x87 error pointers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXRSTOR, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXRSTOR instruction. If software running in 64-bit mode executes an FXRSTOR with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXRSTOR with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, the saved image of XMM0–XMM7 and MXCSR is not loaded into the processor. A general-protection exception occurs if there is an attempt to load a non-zero value to the bits in MXCSR that are defined as reserved and cleared (bits 31-16 and bit 6).

Related Instructions

FWAIT, FXSAVE

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

FXSAVE Save XMM, MMX, and x87 State

The FXSAVE instruction saves the XMM, MMX, and x87 state. A memory location that is not aligned on a 16-byte boundary causes a general-protection exception, or, in some cases, an alignment-check exception.

Mnemonic	Opcode	Description
FXSAVE <i>mem512env</i>	0F AE /0	Saves XMM, MMX, and x87 state in 512-byte memory location.

Unlike FSAVE and FNSAVE, FXSAVE does not alter the x87 tag bits. The contents of the saved MMX/x87 data registers are retained, thus indicating that the registers may be valid (or whatever other value the x87 tag bits indicated prior to the save). To invalidate the contents of the MMX/x87 data registers after FXSAVE, software must execute an FINIT instruction. Also, FXSAVE (like FNSAVE) does not check for pending unmasked x87 floating-point exceptions. An FWAIT instruction can be used for this purpose.

FXSAVE executes faster than FSAVE or FNSAVE because it does not save the x87 pointer registers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXSAVE, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXSAVE instruction. If software running in 64-bit mode executes an FXSAVE with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXSAVE with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, FXSAVE does not save the image of XMM0–XMM7 and MXCSR. For details about the CR4.OSFXSR bit, see “FXSAVE/FXRSTOR Support (OSFXSR) Bit” on page 63 of Volume 2.

Related Instructions

FINIT, FNSAVE, FRSTOR, FSAVE, FXRSTOR, LDMXCSR, STMXCSR

rFLAGS Affected

None.

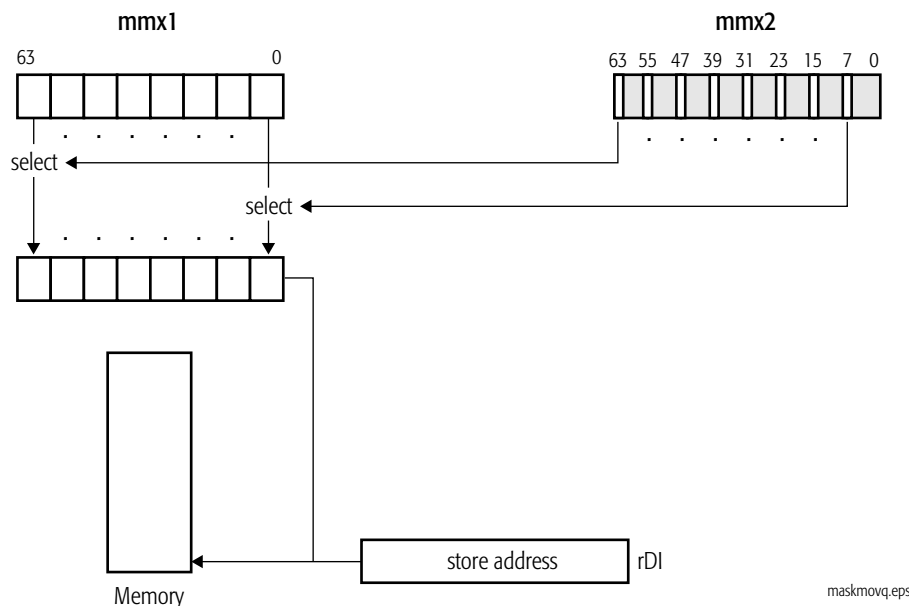
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

MASKMOVQ**Masked Move Quadword**

The MASKMOVQ instruction stores bytes from the first source operand, as selected by the second source operand, to a memory location specified in the rDI and DS registers (except that DS is ignored in 64-bit mode). The first source operand is an MMX register, and the second source operand is another MMX register. The most-significant bit (msb) of each byte in the second source operand specifies the store (1 = store, 0 = no store) of the corresponding byte of the first source operand. The size of the store is determined by the address-size attribute.

Mnemonic	Opcode	Description
MASKMOVQ <i>mmx1, mmx2</i>	0F F7 /r	Store bytes from an MMX register, selected by the most-significant bit of the corresponding byte in another MMX register, to a memory location.



A mask value of all 0s results in the following behavior:

- No data is written to memory.
- Data breakpoints are not guaranteed to be signalled in all implementations (although code breakpoints are guaranteed).
- Page faults and some exceptions associated with memory addressing and are not guaranteed to be signalled in all implementations.
- Illegal addresses in segment registers are signalled, as are segment overrun excep-

tions in real and virtual-86 modes.

MASKMOVQ implicitly uses weakly-ordered, write-combining buffering for the data, as described in Section 7.4, “Buffering and Combining Memory Writes”, on page 215 of Volume 2. If the stored data is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (refer to Section 6.6, “Cache and TLB Management”, on page 201 of Volume 2).

Related Instructions

MASKMOVDQU

Exceptions

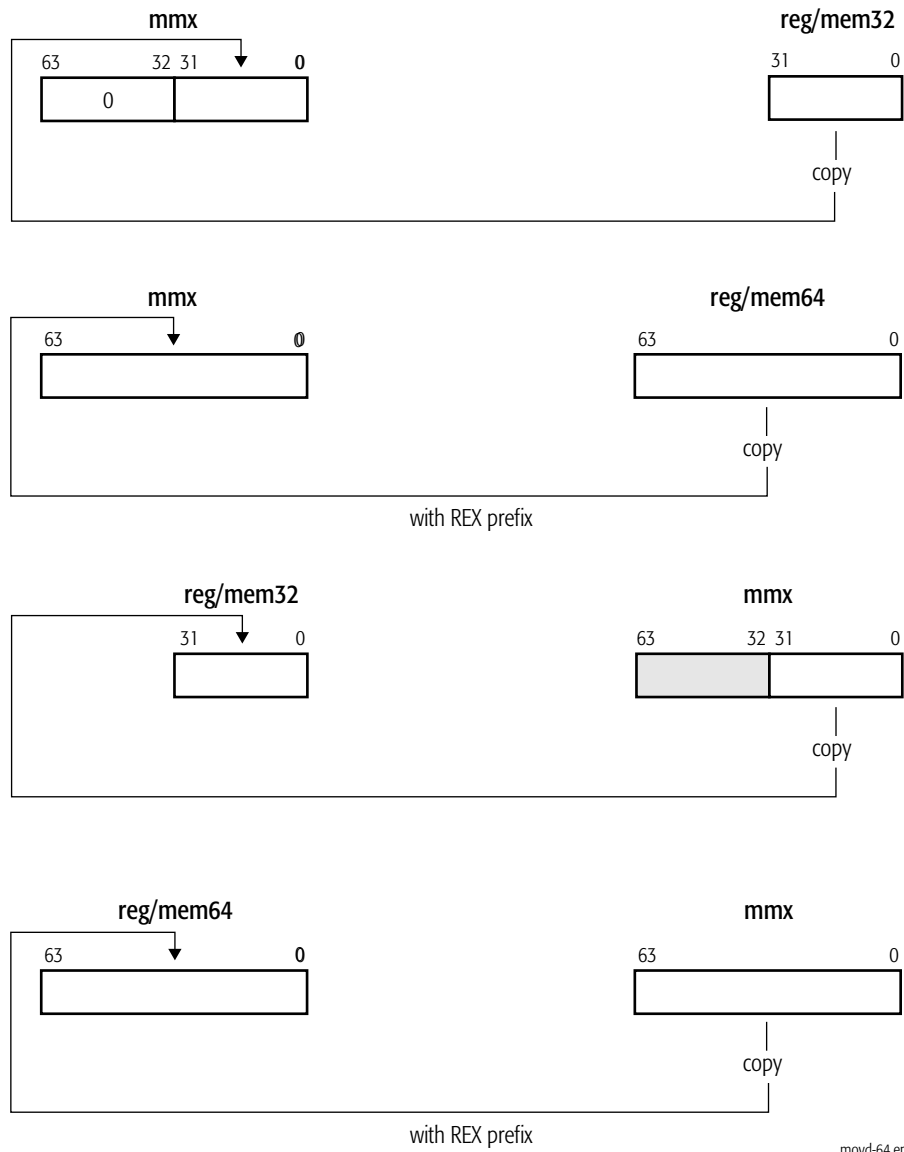
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVD Move Doubleword or Quadword

The MOVD instruction moves a 32-bit or 64-bit value:

- from a 32-bit or 64-bit general-purpose register or memory location to the low-order 32 bits (with zero-extension to 64 bits) or the full 64 bits of an MMX register, or
- from the low-order 32 or the full 64 bits of an MMX register to a 32-bit or 64-bit general-purpose register or memory location.

Mnemonic	Opcode	Description
MOVD <i>mmx, reg/mem32</i>	0F 6E/r	Moves 32-bit value from a general-purpose register or 32-bit memory location to an MMX register.
MOVD <i>mmx, reg/mem64</i>	0F 6E/r	Moves 64-bit value from a general-purpose register or 64-bit memory location to an MMX register.
MOVD <i>reg/mem32, mmx</i>	0F 7E/r	Moves 32-bit value from an MMX register to a general-purpose register or 32-bit memory location.
MOVD <i>reg/mem64, mmx</i>	0F 7E/r	Moves 64-bit value from an MMX register to a general-purpose register or 64-bit memory location.



Related Instructions

MOVDQA, MOVDQU, MOVDQ2Q, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

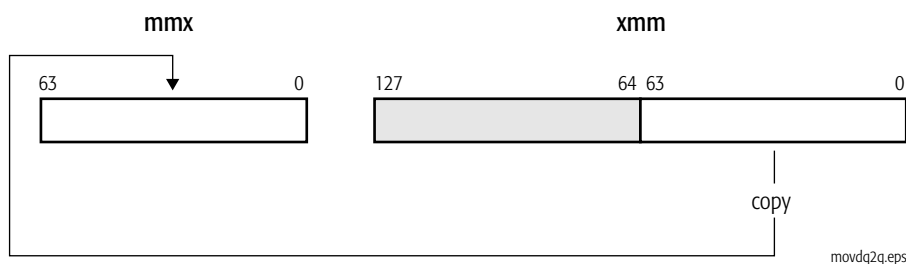
Exceptions

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the EFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVDQ2Q**Move Quadword to Quadword**

The MOVDQ2Q instruction moves the low-order 64-bit value in an XMM register to a 64-bit MMX register.

Mnemonic	Opcode	Description
MOVDQ2Q <i>mmx, xmm</i>	F2 0F D6	Moves low-order 64-bit value from an XMM register to the destination MMX register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVQ, MOVQ2DQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

MOVNTQ**Move Non-Temporal Quadword**

The MOVNTQ instruction stores a 64-bit MMX register value into a 64-bit memory location. This instruction indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining memory write, which minimizes cache pollution. The exact method by which cache pollution is minimized depends on the hardware implementation of the instruction. For further information, see Section 3.9, “Memory Optimization”, on page 115 of Volume 1.

Mnemonic	Opcode	Description
MOVNTQ <i>mem64</i> , mmx	0F E7 /r	Stores a 64-bit MMX register value into a 64-bit memory location, minimizing cache pollution.



MOVNTQ is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE instruction to force strong memory ordering of MOVNTQ with respect to other stores.

MOVNTQ implicitly uses weakly-ordered, write-combining buffering for the data, as described in Section 7.4, “Buffering and Combining Memory Writes”, on page 215 of Volume 2. For data that is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (refer to Section 6.6, “Cache and TLB Management”, on page 201 of Volume 2).

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPD, MOVNTPS

Exceptions

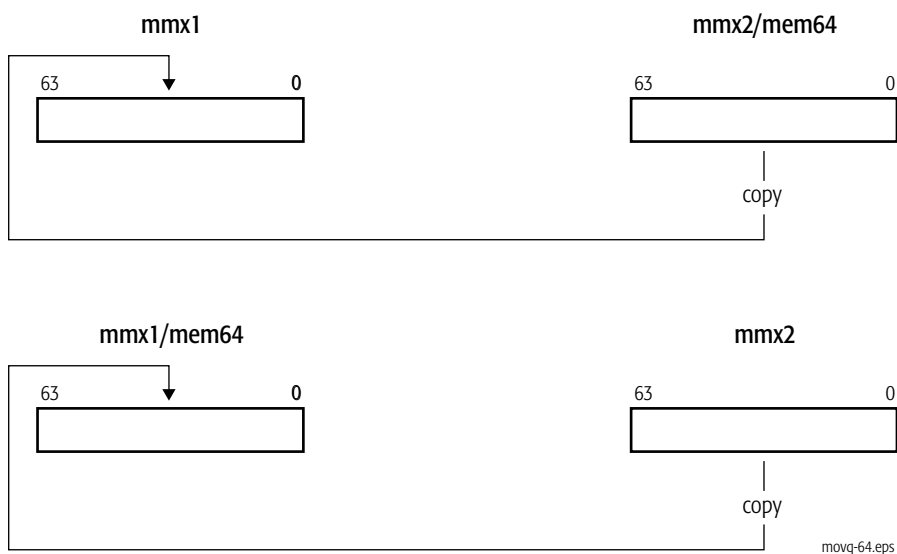
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVQ Move Quadword

The MOVQ instruction moves a 64-bit value:

- from an MMX register or 64-bit memory location to another MMX register, or
- from an MMX register to another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
MOVQ <i>mmx1</i> , <i>mmx2/mem64</i>	0F 6F /r	Moves 64-bit value from an MMX register or memory location to an MMX register.
MOVQ <i>mmx1/mem64</i> , <i>mmx2</i>	0F 7F /r	Moves 64-bit value from an MMX register to an MMX register or memory location.



Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ2DQ

rFLAGS Affected

None.

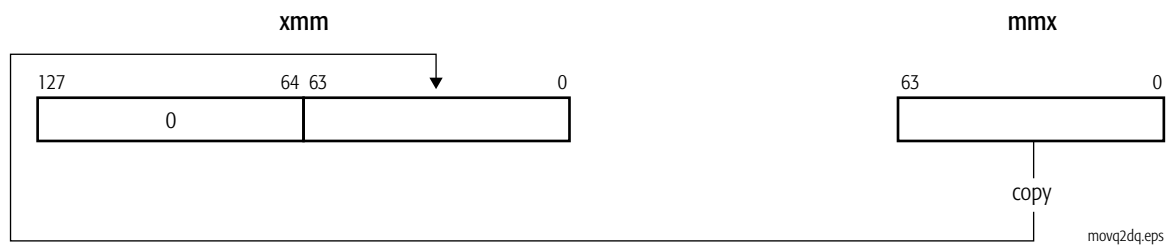
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

MOVQ2DQ**Move Quadword to Quadword**

The MOVQ2DQ instruction moves a 64-bit value from an MMX register to the low-order 64 bits of an XMM register, with zero-extension to 128 bits.

Mnemonic	Opcode	Description
MOVQ2DQ <i>xmm, mmx</i>	F3 0F D6	Moves 64-bit value from an MMX register to an XMM register.



Execution of this instruction causes all fields in the x87 tag word to be set according to their corresponding data, the top-of-stack-pointer bit (TOP) in the x87 status word to be cleared to 0, and any pending x87 exceptions are handled before this instruction is executed. For details, see Section 5.11, “Actions Taken on Executing 64-Bit Media Instructions”, on page 280 of Volume 1.

Related Instructions

MOVD, MOVDQA, MOVDQU, MOVDQ2Q, MOVQ

rFLAGS Affected

None.

MXCSR Flags Affected

None.

Exceptions

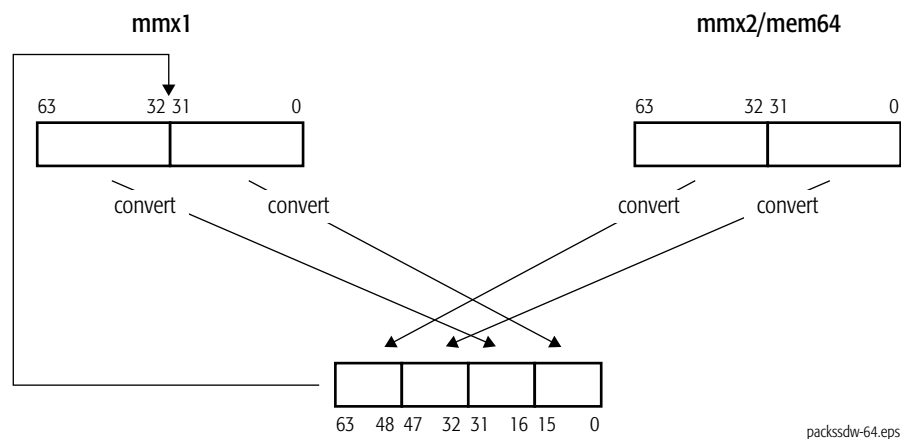
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

PACKSSDW**Pack with Saturation Signed Doubleword to Word**

The PACKSSDW instruction converts each 32-bit signed integer in the first and second source operands to a 16-bit signed integer and packs the converted values into words in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order words of the destination, and the converted values from the second source operand are packed into the high-order words of the destination.

Mnemonic	Opcode	Description
PACKSSDW <i>mmx1, mmx2/mem64</i>	0F 6B /r	Packs 32-bit signed integers in an MMX register and another MMX register or 64-bit memory location into 16-bit signed integers in an MMX register.



For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PACKSSWB, PACKUSWB

rFLAGS Affected

None.

Exceptions

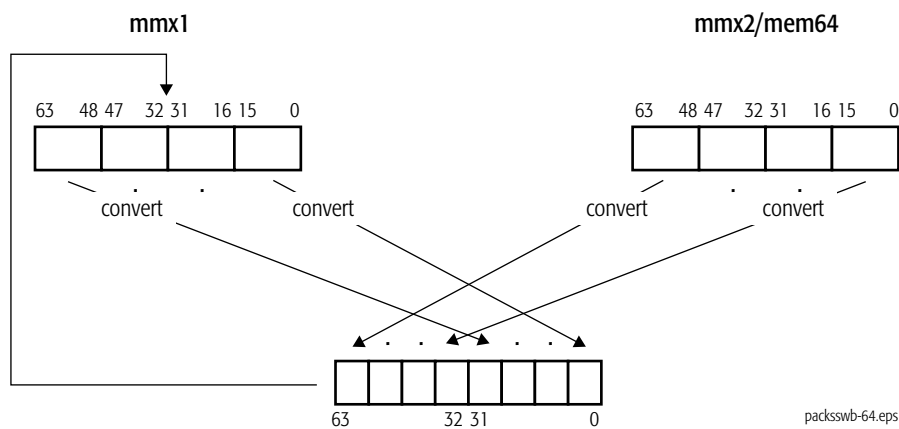
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PACKSSWB**Pack with Saturation Signed Word to Byte**

The PACKSSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit signed integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKSSWB <i>mmx1, mmx2/mem64</i>	OF 63/r	Packs 16-bit signed integers in an MMX register and another MMX register or 64-bit memory location into 8-bit signed integers in an MMX register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PACKSSDW, PACKUSWB

rFLAGS Affected

None.

Exceptions

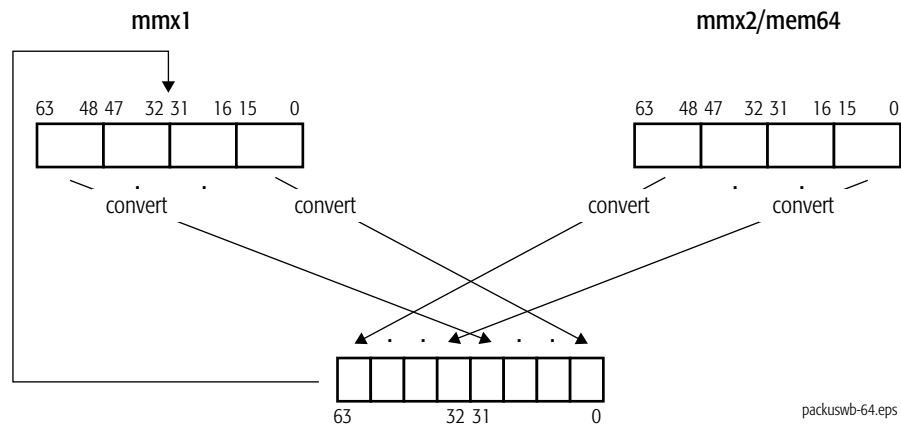
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PACKUSWB**Pack with Saturation Signed Word to Unsigned Byte**

The PACKUSWB instruction converts each 16-bit signed integer in the first and second source operands to an 8-bit unsigned integer and packs the converted values into bytes in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Converted values from the first source operand are packed into the low-order bytes of the destination, and the converted values from the second source operand are packed into the high-order bytes of the destination.

Mnemonic	Opcode	Description
PACKUSWB <i>mmx1, mmx2/mem64</i>	0F 67 /r	Packs 16-bit signed integers in an MMX register and another MMX register or 64-bit memory location into 8-bit unsigned integers in an MMX register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PACKSSDW, PACKSSWB

rFLAGS Affected

None.

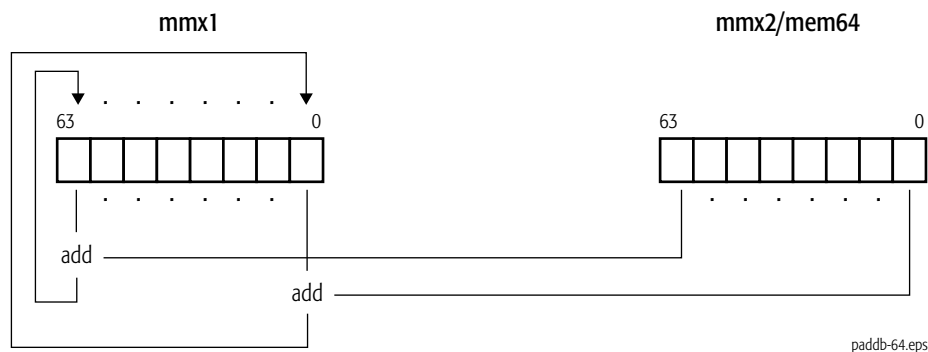
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDB**Packed Add Bytes**

The PADDB instruction adds each packed 8-bit integer value in the first source operand to the corresponding packed 8-bit integer in the second source operand and writes the integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDB <i>mmx1, mmx2/mem64</i>	0F FC /r	Adds packed byte integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

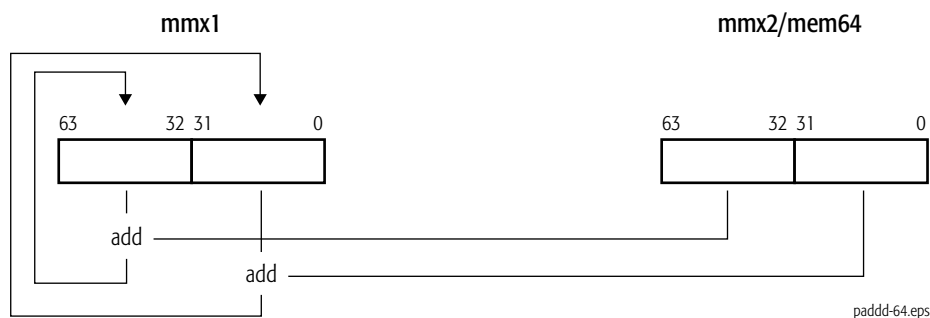
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADD**Packed Add Doublewords**

The PADD instruction adds each packed 32-bit integer value in the first source operand to the corresponding packed 32-bit integer in the second source operand and writes the integer result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADD <i>mmx1, mmx2/mem64</i>	0F FE /r	Adds packed 32-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PADDB, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDQ**Packed Add Quadwords**

The PADDQ instruction adds each packed 64-bit integer value in the first source operand to the corresponding packed 64-bit integer in the second source operand and writes the integer result of each addition in the corresponding quadword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDQ <i>mmx1, mmx2/mem64</i>	0F D4 /r	Adds 64-bit integer value in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PADDB, PADDD, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

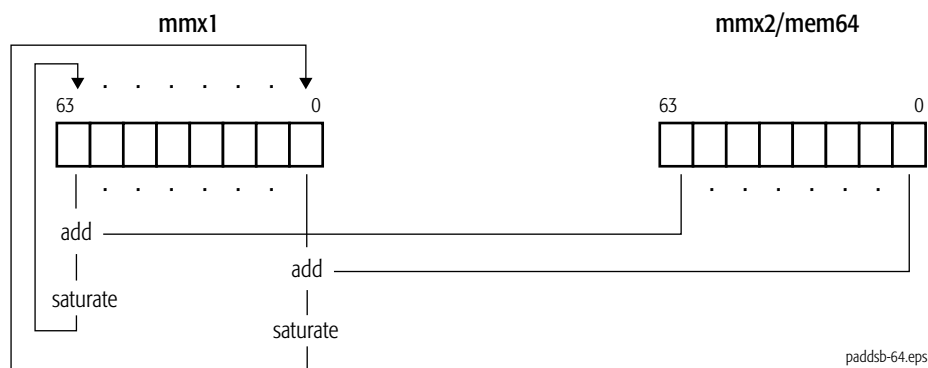
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDSB**Packed Add Signed with Saturation Bytes**

The PADDSB instruction adds each packed 8-bit signed integer value in the first source operand to the corresponding packed 8-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDSB <i>mmx1, mmx2/mem64</i>	0F EC /r	Adds packed byte signed integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest representable signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSW, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

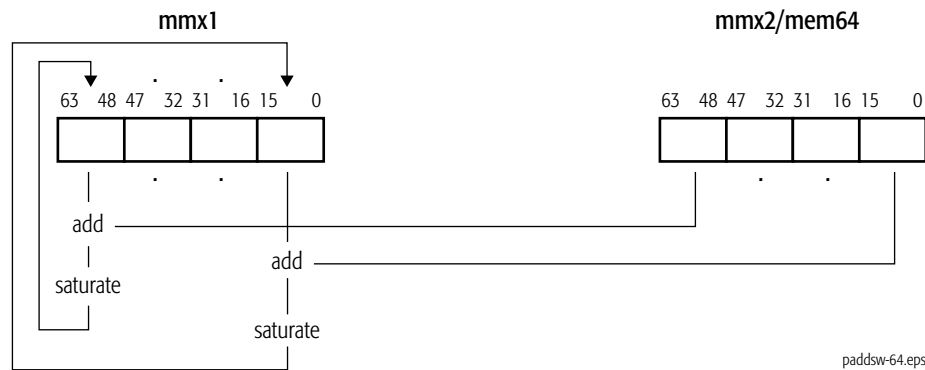
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDSW**Packed Add Signed with Saturation Words**

The PADDSW instruction adds each packed 16-bit signed integer value in the first source operand to the corresponding packed 16-bit signed integer in the second source operand and writes the signed integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDSW <i>mmx1, mmx2/mem64</i>	0F ED /r	Adds packed 16-bit signed integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest representable signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDUSB, PADDUSW, PADDW

rFLAGS Affected

None.

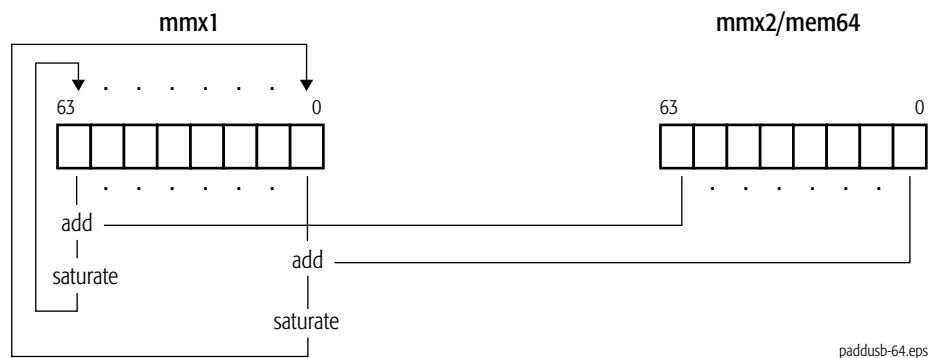
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDUSB**Packed Add Unsigned with Saturation Bytes**

The PADDUSB instruction adds each packed 8-bit unsigned integer value in the first source operand to the corresponding packed 8-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDUSB <i>mmx1, mmx2/mem64</i>	0F DC /r	Adds packed byte unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSW, PADDW

rFLAGS Affected

None.

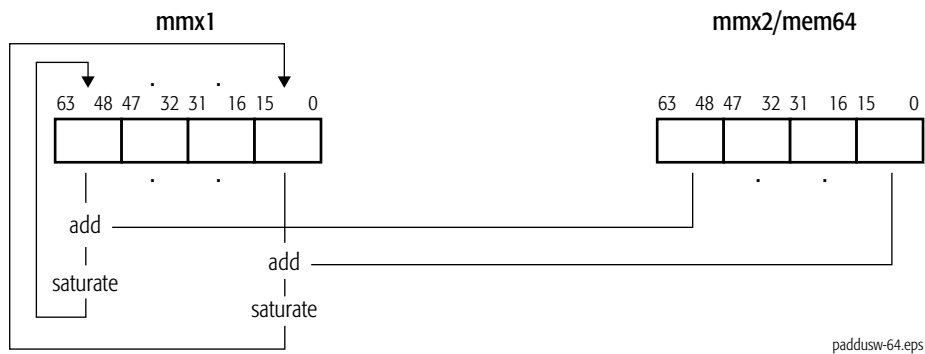
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDUSW**Packed Add Unsigned with Saturation Words**

The PADDUSW instruction adds each packed 16-bit unsigned integer value in the first source operand to the corresponding packed 16-bit unsigned integer in the second source operand and writes the unsigned integer result of each addition in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDUSW <i>mmx1, mmx2/mem64</i>	0F DD /r	Adds packed 16-bit unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDW

rFLAGS Affected

None.

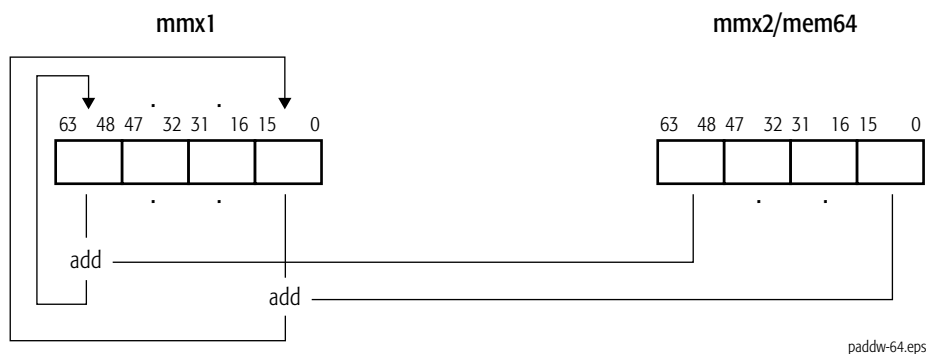
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PADDW Packed Add Words

The PADDW instruction adds each packed 16-bit integer value in the first source operand to the corresponding packed 16-bit integer in the second source operand and writes the integer result of each addition in the corresponding word of the destination (second source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PADDW <i>mmx1, mmx2/mem64</i>	0F FD /r	Adds packed 16-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PADDB, PADDD, PADDQ, PADDSB, PADDSW, PADDUSB, PADDUSW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PAND**Packed Logical Bitwise AND**

The PAND instruction performs a bitwise logical AND of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAND <i>mmx1, mmx2/mem64</i>	0F DB /r	Performs bitwise logical AND of values in an MMX register and in another MMX register or 64-bit memory location and writes the result in the destination MMX register.

**Related Instructions**

PANDN, POR, PXOR

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PANDN**Packed Logical Bitwise AND NOT**

The PANDN instruction performs a bitwise logical AND of the value in the second source operand and the one's complement of the value in the first source operand and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PANDN <i>mmx1, mmx2/mem64</i>	0F DF /r	Performs bitwise logical AND NOT of values in an MMX register and in another MMX register or 64-bit memory location and writes the result in the destination MMX register.

**Related Instructions**

PAND, POR, PXOR

rFLAGS Affected

None.

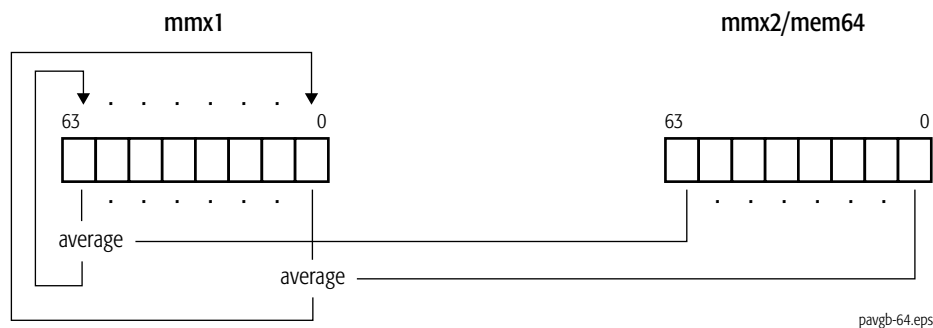
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PAVGB**Packed Average Unsigned Bytes**

The PAVGB instruction computes the rounded average of each packed unsigned 8-bit integer value in the first source operand and the corresponding packed 8-bit unsigned integer in the second source operand and writes each average in the corresponding byte of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 9-bit temporary sum, and then right-shifting the temporary sum by one bit position. The destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGB <i>mmx1, mmx2/mem64</i>	0F E0/r	Averages packed 8-bit unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.

**Related Instructions**

PAVGW

rFLAGS Affected

None.

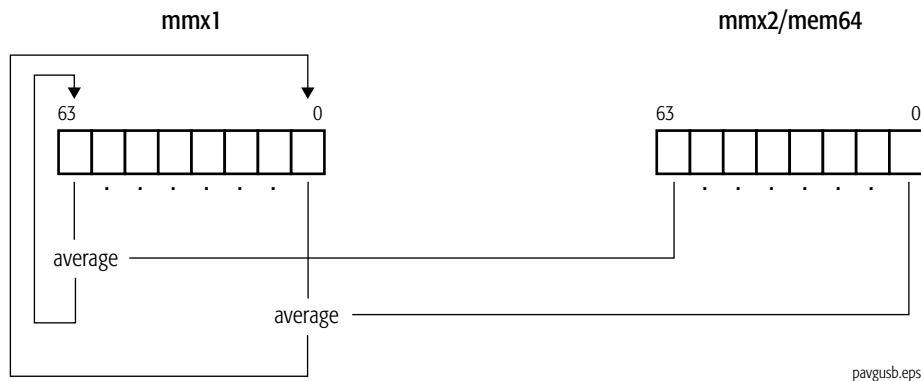
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PAVGUSB**Packed Average Unsigned Bytes**

The PAVGUSB instruction computes the rounded-up average of each packed unsigned 8-bit integer value in the first source operand and the corresponding packed 8-bit unsigned integer in the second source operand and writes each average in the corresponding byte of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 9-bit temporary sum, and then right-shifting the temporary sum by one bit position. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGUSB <i>mmx1, mmx2/mem64</i>	0F 0F BF	Averages packed 8-bit unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.



The PAVGUSB instruction performs a function identical to the 64-bit version of the PAVGB instruction, although the two instructions have different opcodes. PAVGUSB is a 3DNow! instruction. It is useful for pixel averaging in MPEG-2 motion compensation and video scaling operations.

Related Instructions

None.

rFLAGS Affected

None.

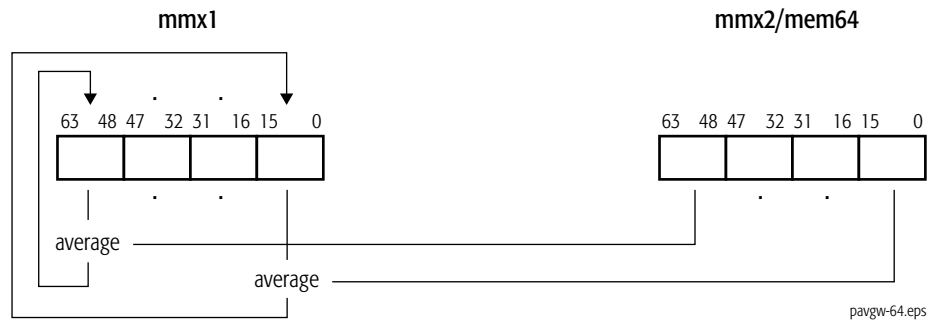
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PAVGW**Packed Average Unsigned Words**

The PAVGW instruction computes the rounded average of each packed unsigned 16-bit integer value in the first source operand and the corresponding packed 16-bit unsigned integer in the second source operand and writes each average in the corresponding word of the destination (first source). The average is computed by adding each pair of operands, adding 1 to the 17-bit temporary sum, and then right-shifting the temporary sum by one bit position. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PAVGW <i>mmx1, mmx2/mem64</i>	0F E3 /r	Averages packed 16-bit unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes the result in the destination MMX register.

**Related Instructions**

PAVGB

rFLAGS Affected

None.

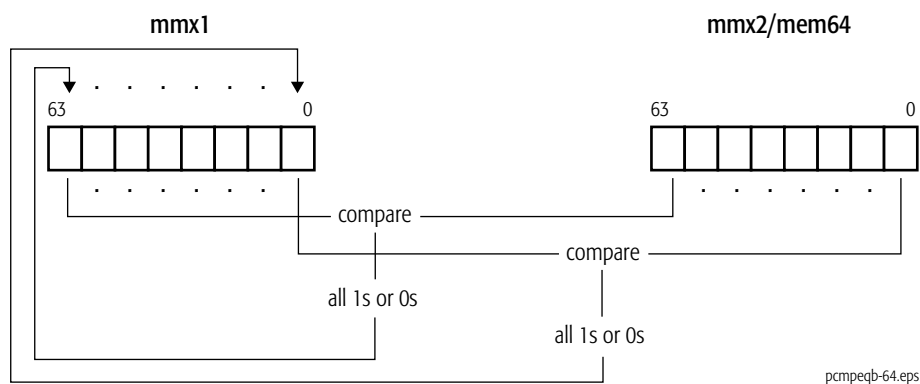
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPEQB**Packed Compare Equal Bytes**

The PCMPEQB instruction compares corresponding packed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQB <i>mmx1, mmx2/mem64</i>	0F 74 /r	Compares packed bytes in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

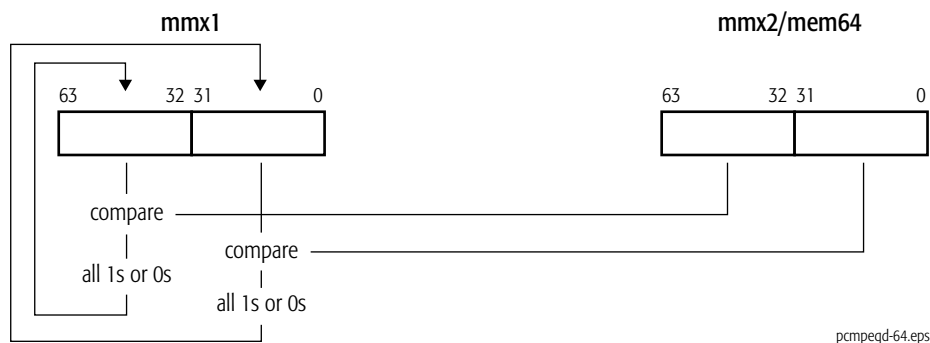
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPEQD**Packed Compare Equal Doublewords**

The PCMPEQD instruction compares corresponding packed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQD <i>mmx1, mmx2/mem64</i>	0F 76 /r	Compares packed doublewords in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQB, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

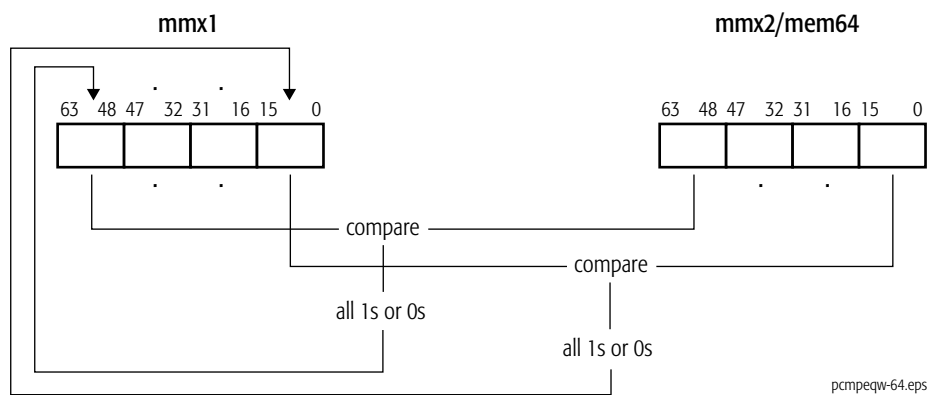
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPEQW**Packed Compare Equal Words**

The PCMPEQW instruction compares corresponding packed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPEQW <i>mmx1, mmx2/mem64</i>	0F 75 /r	Compares packed 16-bit values in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQB, PCMPEQD, PCMPGTB, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

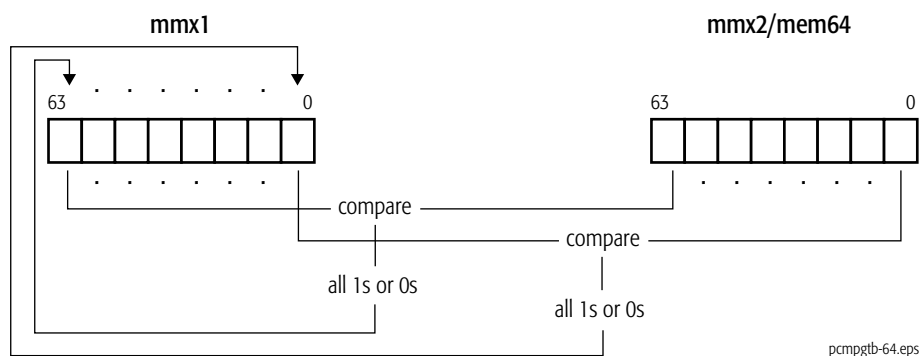
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPGTB**Packed Compare Greater Than Signed Bytes**

The PCMPGTB instruction compares corresponding packed signed bytes in the first and second source operands and writes the result of each compare in the corresponding byte of the destination (first source). For each pair of bytes, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTB <i>mmx1, mmx2/mem64</i>	0F 64 /r	Compares packed signed bytes in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTD, PCMPGTW

rFLAGS Affected

None.

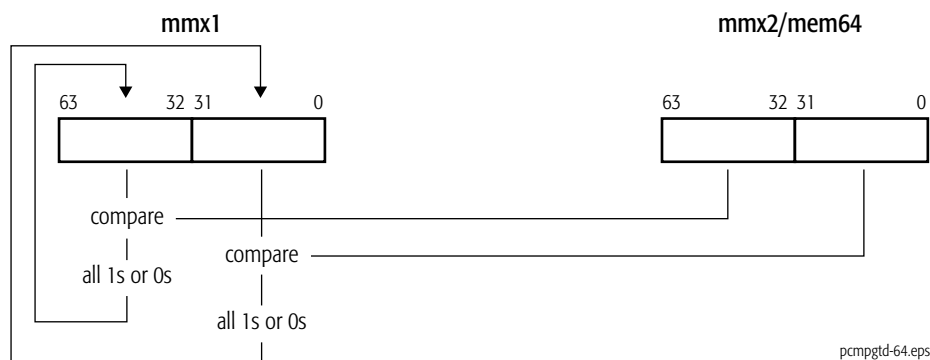
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPGTD**Packed Compare Greater Than Signed Doublewords**

The PCMPGTD instruction compares corresponding packed signed 32-bit values in the first and second source operands and writes the result of each compare in the corresponding 32 bits of the destination (first source). For each pair of doublewords, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTD <i>mmx1, mmx2/mem64</i>	0F 66/r	Compares packed signed 32-bit values in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTW

rFLAGS Affected

None.

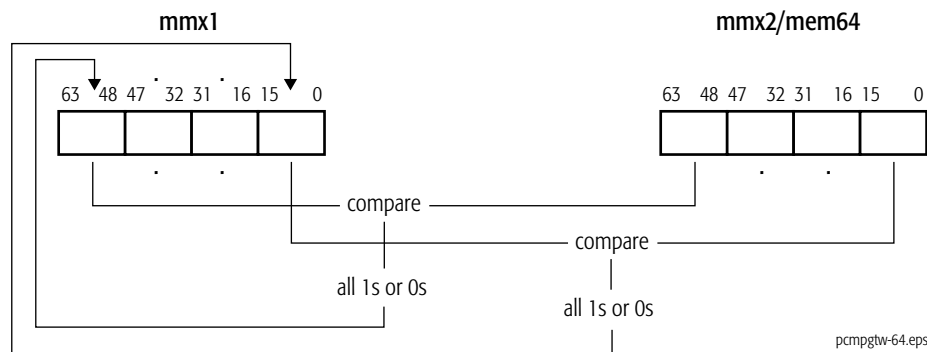
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PCMPGTW**Packed Compare Greater Than Signed Words**

The PCMPGTW instruction compares corresponding packed signed 16-bit values in the first and second source operands and writes the result of each compare in the corresponding 16 bits of the destination (first source). For each pair of words, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PCMPGTW <i>mmx1, mmx2/mem64</i>	0F 65 /r	Compares packed signed 16-bit values in an MMX register and an MMX register or 64-bit memory location.

**Related Instructions**

PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PEXTRW Extract Packed Word

The PEXTRW instruction extracts a 16-bit value from an MMX register, as selected by the immediate byte operand (as shown in Table 2-13) and writes it to the low-order word of a 32-bit or 64-bit general-purpose register, with zero-extension to 32 or 64 bits.

Mnemonic	Opcode	Description
PEXTRW <i>reg32/64, mmx, imm8</i>	0F C5 /r ib	Extracts a 16-bit value from an MMX register and writes it to low-order 16 bits of a general-purpose register.

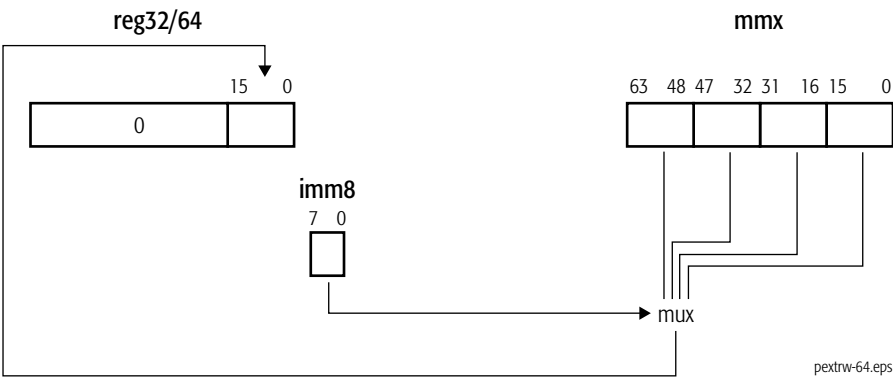


Table 2-13. Immediate-Byte Operand Encoding for 64-Bit PEXTRW

Immediate-Byte Bit Field	Value of Bit Field	Source Bits Extracted
1:0	0	15:0
	1	31:16
	2	47:32
	3	63:48

Related Instructions

PINSRW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PF2ID

Packed Floating-Point to Integer Doubleword Conversion

The PF2ID instruction converts two packed single-precision floating-point values in an MMX register or a 64-bit memory location to two packed 32-bit signed integer values and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). The numeric range for source and destination operands is shown in Table 2-14.

Mnemonic	Opcode	Description
PF2ID <i>mmx1, mmx2/mem64</i>	0F 0F 1D	Converts packed single-precision floating-point values in an MMX register or memory location to a doubleword integer value in the destination MMX register.

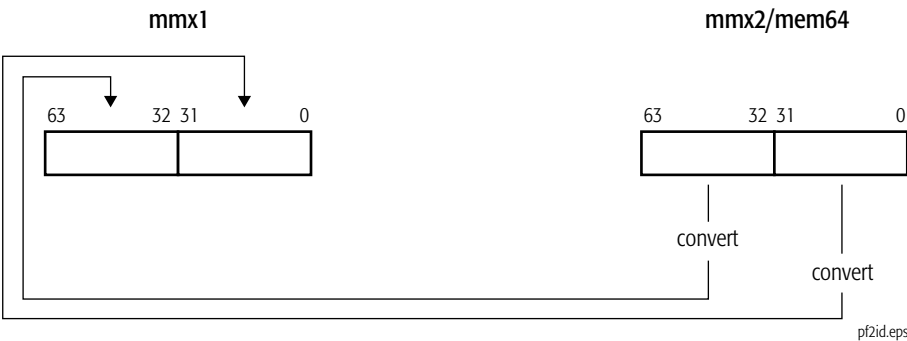


Table 2-14. Numeric Range for PF2ID Results

Source 2	Source 1 and Destination
0	0
Normal, $\text{abs}(\text{Source 1}) < 1$	0
Normal, $-2^{31} < \text{Source 1} \leq -1$	Round to zero (Source 1)
Normal, $1 \leq \text{Source 1} < 2^{31}$	Round to zero (Source 1)
Normal, $\text{Source 1} \geq 2^{31}$	7FFF_FFFFh
Normal, $\text{Source 1} \leq -2^{31}$	8000_0000h
Unsupported	Undefined

Related Instructions

PF2IW, PI2FD, PI2FW

rFLAGS Affected

None.

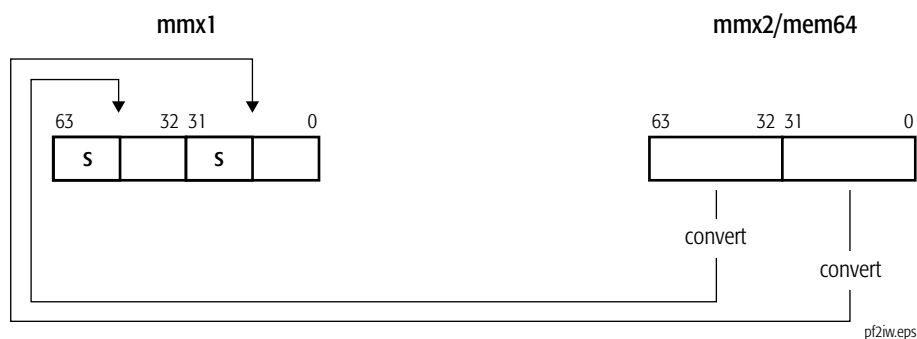
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PF2IW**Packed Floating-Point to Integer Word Conversion**

The PF2IW instruction converts two packed single-precision floating-point values in an MMX register or a 64-bit memory location to two packed 16-bit signed integer values, sign-extended to 32-bits, and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero). The numeric range for source and destination operands is shown in Table 2-15. Arguments outside the range representable by signed 16-bit integers are saturated to the largest and smallest 16-bit integer, depending on their sign.

Mnemonic	Opcode	Description
PF2IW <i>mmx1, mmx2/mem64</i>	0F 0F 1C	Converts packed single-precision floating-point values in an MMX register or memory location to word integer values in the destination MMX register.

**Table 2-15. Numeric Range for PF2IW Results**

Source 2	Source 1 and Destination
0	0
Normal, $\text{abs}(\text{Source 1}) < 1$	0
Normal, $-2^{15} < \text{Source 1} \leq -1$	Round to zero (Source 1)
Normal, $1 \leq \text{Source 1} < 2^{15}$	Round to zero (Source 1)

Table 2-15. Numeric Range for PF2IW Results

Source 2	Source 1 and Destination
Normal, Source 1 $\geq 2^{15}$	7FFF_FFFFh
Normal, Source 1 $\leq 2^{15}$	8000_0000h
Unsupported	Undefined

Related Instructions

PF2ID, PI2FD, PI2FW

rFLAGS Affected

None.

Exceptions

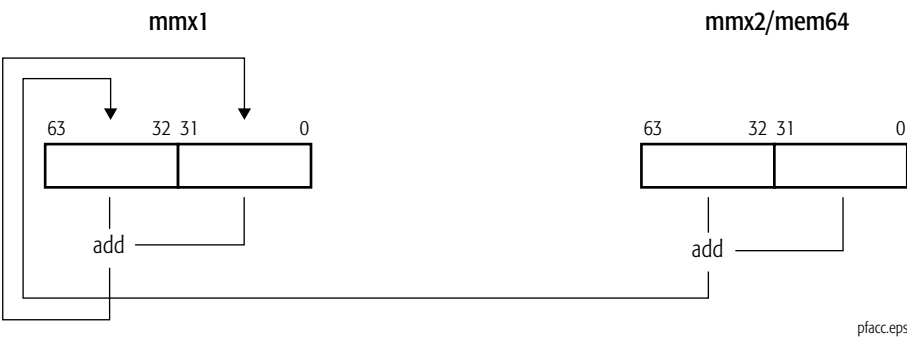
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD Extensions to 3DNow!™ bit (bit 30) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFACC

Packed Floating-Point Accumulate

The PFACC instruction adds the two single-precision floating-point values in the first source operand and adds the two single-precision values in the second source operand and writes the two results to the low-order and high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFACC <i>mmx1, mmx2/mem64</i>	0F 0F AE	Accumulates packed single-precision floating-point values in an MMX register or 64-bit memory location and another MMX register and writes each result in the destination MMX register.



The numeric range for operands is shown in Table 2-16.

Table 2-16. Numeric Range for PFACC Results

Source Operand		High Operand ²		
		0	Normal	Unsupported
Low Operand ¹	0	+/- 0 ³	High Operand	High Operand
	Normal	Low Operand	Normal, +/- 0 ⁴	Undefined
	Unsupported ⁵	Low Operand	Undefined	Undefined

Note:

1. Least-significant floating-point value in first or second source operand.
2. Most-significant floating-point value in first or second source operand.
3. The sign of the result is the logical AND of the signs of the low and high operands
4. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero with the sign of the operand (low or high) that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of the low operand.
5. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFADD, PFNACC, PFPNACC

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFADD**Packed Floating-Point Add**

The PFADD instruction adds each packed single-precision floating-point value in the first source operand to the corresponding packed single-precision floating-point value in the second operand and writes the result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-17.

Mnemonic	Opcode	Description
PFADD <i>mmx1, mmx2/mem64</i>	0F 0F 9E	Adds two packed single-precision floating-point values in an MMX register or 64-bit memory location and another MMX register and writes each result in the destination MMX register.

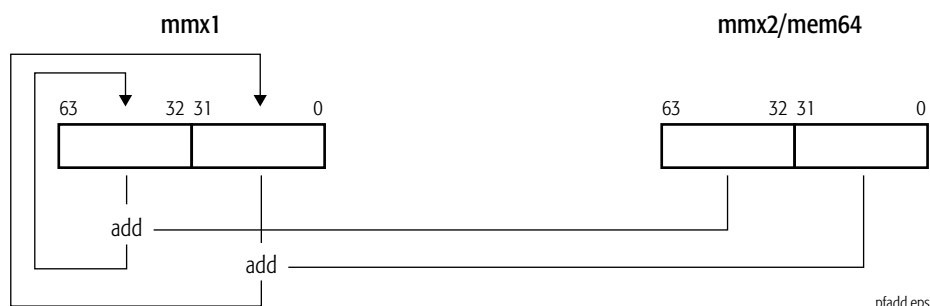


Table 2-17. Numeric Range for the PFADD Results

Source Operand		Most-Significant Doubleword		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	Source 2	Source 2
	Normal	Source 1	Normal, +/- 0 ²	Undefined
	Unsupported ³	Source 1	Undefined	Undefined

Note:

1. The sign of the result is the logical AND of the signs of the source operands
2. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero with the sign of the source operand that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of source 1. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of source 1.
3. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFACC, PFNACC, PFPNACC

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFCMPEQ**Packed Floating-Point Compare Equal**

The PFCMPEQ instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the values are equal, the result is all 1s. If the values are not equal, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-18.

Mnemonic	Opcode	Description
PFCMPEQ <i>mmx1, mmx2/mem64</i>	0F 0F B0	Compares two pairs of packed single-precision floating-point values in an MMX register and an MMX register or 64-bit memory location.

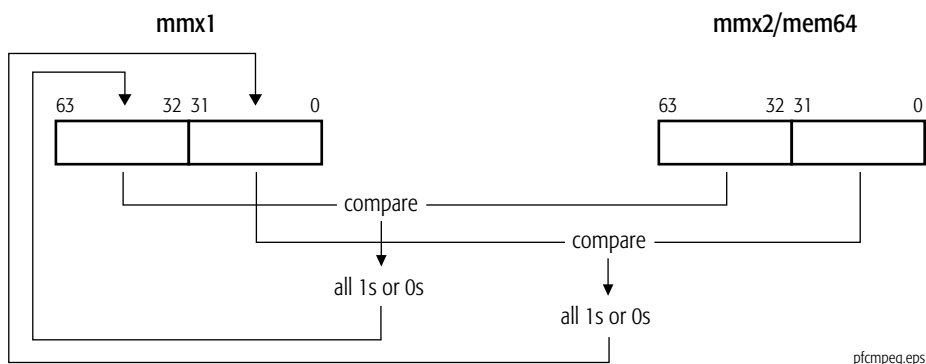


Table 2-18. Numeric Range for the PFCMPEQ Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	FFFF_FFFFh ¹	0000_0000h	0000_0000h
	Normal	0000_0000h	0000_0000h or FFFF_FFFFh ²	0000_0000h
	Unsupported ³	0000_0000h	0000_0000h	Undefined

Note:

1. Positive zero is equal to negative zero.
2. The result is FFFF_FFFFh if source 1 and source 2 have identical signs, exponents, and mantissas. Otherwise, the result is 0000_0000h.
3. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFCMPGE, PFCMPGT

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFCMPGE**Packed Floating-Point Compare Greater or Equal**

The PFCMPGE instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the value in the first source operand is greater than or equal to the value in the second source operand, the result is all 1s. If the value in the first source operand is less than the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-19.

Mnemonic	Opcode	Description
PFCMPGE <i>mmx1, mmx2/mem64</i>	0F 0F 90	Compares two pairs of packed single-precision floating-point values in an MMX register and an MMX register or 64-bit memory location.

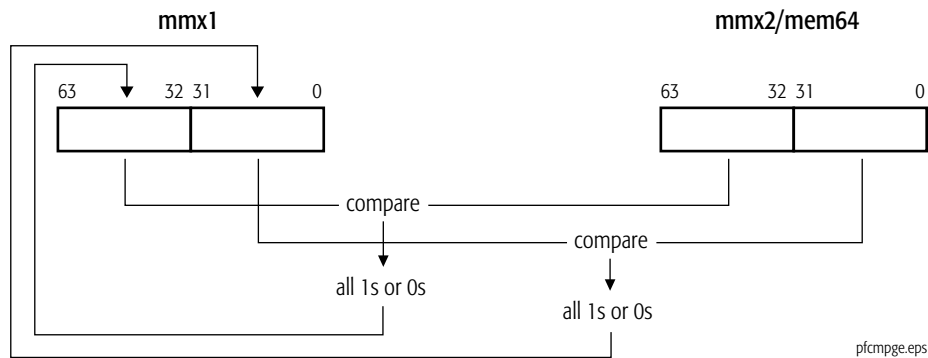


Table 2-19. Numeric Range for the PFCMPGE Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	FFFF_FFFFh ¹	0000_0000h, FFFF_FFFFh ²	Undefined
	Normal	0000_0000h, FFFF_FFFFh ³	0000_0000h, FFFF_FFFFh ⁴	Undefined
	Unsupported ⁵	Undefined	Undefined	Undefined

Note:

1. Positive zero is equal to negative zero.
2. The result is FFFF_FFFFh, if source 2 is negative. Otherwise, the result is 0000_0000h.
3. The result is FFFF_FFFFh, if source 1 is positive. Otherwise, the result is 0000_0000h.
4. The result is FFFF_FFFFh, if source 1 is positive and source 2 is negative, or if they are both negative and source 1 is smaller than or equal in magnitude to source 2, or if source 1 and source 2 are both positive and source 1 is greater than or equal in magnitude to source 2. The result is 0000_0000h in all other cases.
5. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFCMPEQ, PFCMPGT

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFCMPGT**Packed Floating-Point Compare Greater Than**

The PFCMPGT instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the result of each comparison in the corresponding doubleword of the destination (first source). For each pair of floating-point values, if the value in the first source operand is greater than the value in the second source operand, the result is all 1s. If the value in the first source operand is less than or equal to the value in the second source operand, the result is all 0s. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-20.

Mnemonic	Opcode	Description
PFCMPGT <i>mmx1, mmx2/mem64</i>	OF OF A0	Compares two pairs of packed single-precision floating-point values in an MMX register and an MMX register or 64-bit memory location.

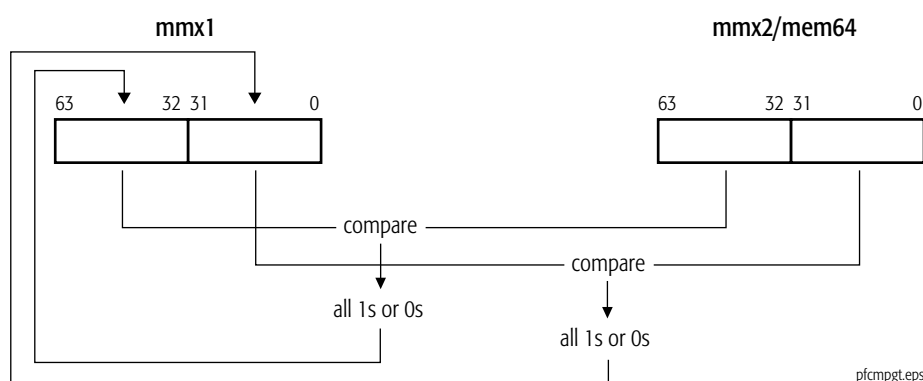


Table 2-20. Numeric Range for the PFCMPGT Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	0000_0000h	0000_0000h, FFFF_FFFFh ¹	Undefined
	Normal	0000_0000h, FFFF_FFFFh ²	0000_0000h, FFFF_FFFFh ³	Undefined
	Unsupported ⁴	Undefined	Undefined	Undefined
Note: <ol style="list-style-type: none"> 1. The result is FFFF_FFFFh, if source 2 is negative. Otherwise, the result is 0000_0000h. 2. The result is FFFF_FFFFh, if source 1 is positive. Otherwise, the result is 0000_0000h. 3. The result is FFFF_FFFFh, if source 1 is positive and source 2 is negative, or if they are both negative and source 1 is smaller in magnitude than source 2, or if source 1 and source 2 are positive and source 1 is greater in magnitude than source 2. The result is 0000_0000h in all other cases. 4. "Unsupported" means that the exponent is all ones (1s). 				

Related Instructions

PFCMPEQ, PFCMPGE

rFLAGS Affected

None.

Exceptions

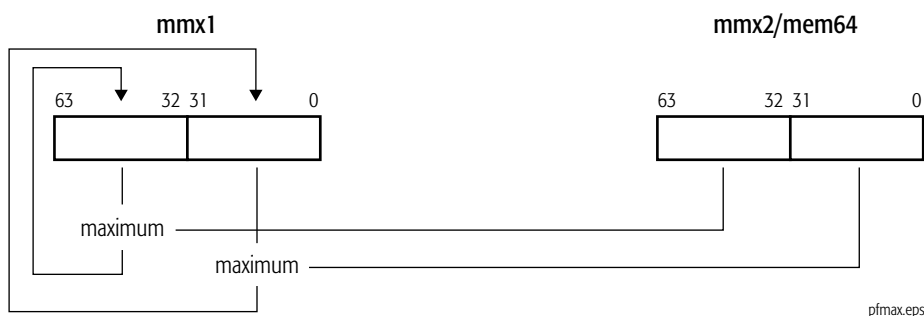
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFMAX**Packed Single-Precision Floating-Point
Maximum**

The PFMAX instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the maximum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFMAX <i>mmx1</i> , <i>mmx2/mem64</i>	0F 0F A4	Compares two pairs of packed single-precision values in an MMX register and another MMX register or 64-bit memory location and writes the maximum value of each comparison in the destination MMX register.



Any operation with a zero and a negative number returns positive zero. An operation consisting of two zeros returns positive zero. If either source operand is an undefined value, the result is undefined. The numeric range for source and destination operands is shown in Table 2-21.

Table 2-21. Numeric Range for the PFMAX Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+0	Source 2, +0 ¹	Undefined
	Normal	Source 1, +0 ²	Source 1/Source 2 ³	Undefined
	Unsupported ⁴	Undefined	Undefined	Undefined
Note: <ol style="list-style-type: none"> 1. The result is source 2, if source 2 is positive. Otherwise, the result is positive zero. 2. The result is source 1, if source 1 is positive. Otherwise, the result is positive zero. 3. The result is source 1, if source 1 is positive and source 2 is negative. The result is source 1, if both are positive and source 1 is greater in magnitude than source 2. The result is source 1, if both are negative and source 1 is lesser in magnitude than source 2. The result is source 2 in all other cases. 4. "Unsupported" means that the exponent is all ones (1s). 				

Related Instructions

PFMIN

rFLAGS Affected

None.

Exceptions

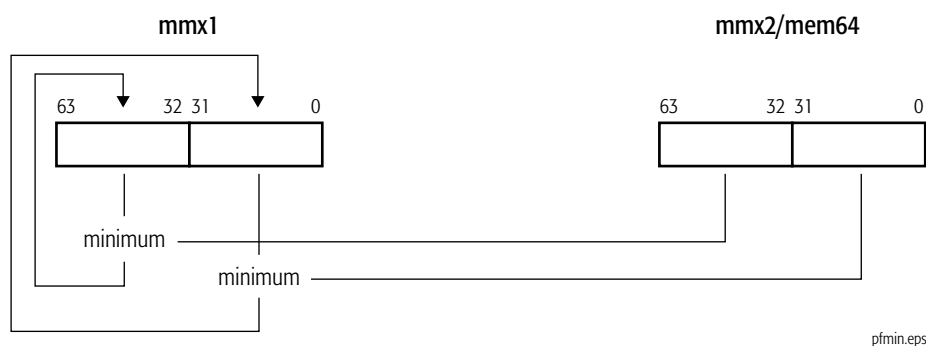
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFCMIN**Packed Single-Precision Floating-Point Minimum**

The PFCMIN instruction compares each of the two packed single-precision floating-point values in the first source operand with the corresponding packed single-precision floating-point value in the second source operand and writes the minimum of the two values for each comparison in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFCMIN <i>mmx1, mmx2/mem64</i>	0F 0F 94	Compares two pairs of packed single-precision values in an MMX register and another MMX register or 64-bit memory location and writes the minimum value of each comparison in the destination MMX register.



Any operation with a zero and a positive number returns positive zero. An operation consisting of two zeros returns positive zero. If either source operand is an undefined value, the result is undefined. The numeric range for source and destination operands is shown in Table 2-22.

Table 2-22. Numeric Range for the PFMIN Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+0	Source 2, +0 ¹	Undefined
	Normal	Source 1, +0 ²	Source 1/Source 2 ³	Undefined
	Unsupported ⁴	Undefined	Undefined	Undefined
Note: <ol style="list-style-type: none"> 1. The result is source 2, if source 2 is negative. Otherwise, the result is positive zero. 2. The result is source 1, if source 1 is negative. Otherwise, the result is positive zero. 3. The result is source 1, if source 1 is negative and source 2 is positive. The result is source 1, if both are negative and source 1 is greater in magnitude than source 2. The result is source 1, if both are positive and source 1 is lesser in magnitude than source 2. The result is source 2 in all other cases. 4. "Unsupported" means that the exponent is all ones (1s). 				

Related Instructions

PFMAX

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFMUL

Packed Floating-Point Multiply

The PFMUL instruction multiplies each of the two packed single-precision floating-point values in the first source operand by the corresponding packed single-precision floating-point value in the second source operand and writes the result of each multiplication in the corresponding doubleword of the destination (first source). The numeric range for source and destination operands is shown in Table 2-23. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFMUL <i>mmx1, mmx2/mem64</i>	0F 0B 4	Multiplies packed single-precision floating-point values in an MMX register and another XMM register or 64-bit memory location and writes the result in the destination MMX register.

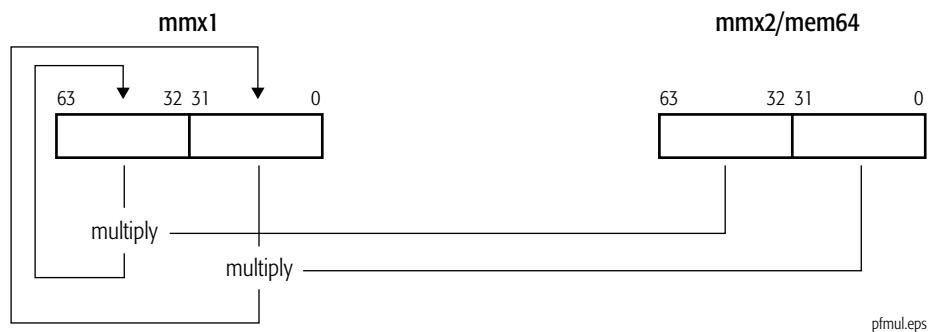


Table 2-23. Numeric Range for the PFMUL Instruction

Operand	Value	Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	+/- 0 ¹	+/- 0 ¹
	Normal	+/- 0 ¹	Normal, +/- 0 ²	Undefined
	Unsupported ³	+/- 0 ¹	Undefined	Undefined

Note:

1. The sign of the result is the exclusive-OR of the signs of the source operands.
2. If the absolute value of the result is less than 2^{-126} , the result is zero with the sign being the exclusive-OR of the signs of the source operands. If the absolute value of the product is greater than or equal to 2^{128} , the result is the largest normal number with the sign being the exclusive-OR of the signs of the source operands.
3. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

None.

rFLAGS Affected

None.

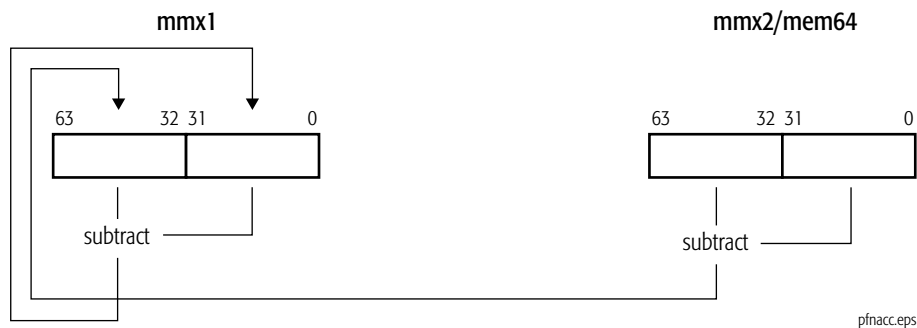
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFNACC**Packed Floating-Point Negative Accumulate**

The PFNACC instruction subtracts the first source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, subtracts the second source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, and writes each result to the low-order or high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFNACC <i>mmx1, mmx2/mem64</i>	0F 0F 8A	Subtracts the packed single-precision floating-point values in an MMX register or 64-bit memory location and another MMX register and writes each value in the destination MMX register.



The numeric range for operands is shown in Table 2-24.

Table 2-24. Numeric Range of PFNACC Results

Source Operand		High Operand ²		
		0	Normal	Unsupported
Low Operand ¹	0	+/- 0 ³	- High Operand	- High Operand
	Normal	Low Operand	Normal, +/- 0 ⁴	Undefined
	Unsupported ⁵	Low Operand	Undefined	Undefined

Note:

1. Least-significant floating-point value in first or second source operand.
2. Most-significant floating-point value in first or second source operand.
3. The sign is the logical AND of the sign of the low operand and the inverse of the sign of the high operand.
4. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the low operand is larger in magnitude than the high operand, the sign of this zero is the same as the sign of the low operand, else it is the inverse of the sign of the high operand. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of the low operand.
5. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFSUB, PFACC, PFPNACC

rFLAGS Affected

None.

Exceptions

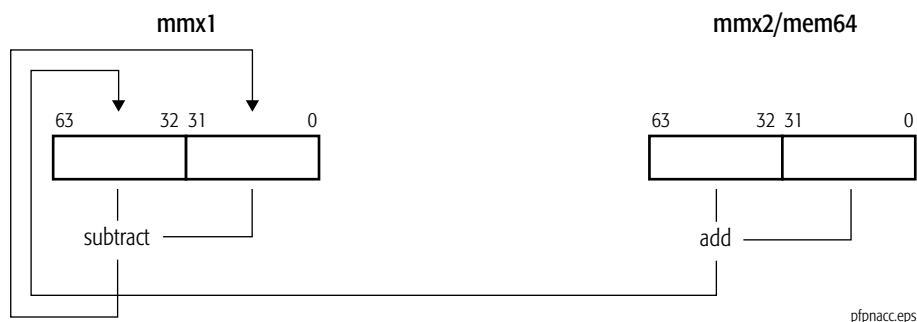
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD Extensions to 3DNow!™ bit (bit 30) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFPNACC**Packed Floating-Point Positive-Negative Accumulate**

The PFPNACC instruction subtracts the first source operand's high-order single-precision floating-point value from its low-order single-precision floating-point value, adds the two single-precision values in the second source operand, and writes each result to the low-order or high-order doubleword, respectively, of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PFPNACC <i>mmx1, mmx2/mem64</i>	0F 0F 8E	Subtracts the packed single-precision floating-point values in an MMX register, adds the packed single-precision floating-point values in another MMX register or 64-bit memory location, and writes each value in the destination MMX register.



The numeric range for operands is shown in Table 2-25 (for the low result) and Table 2-26 (for the high result).

Table 2-25. Numeric Range of PFPNACC Result (Low Result)

Source Operand		High Operand ²		
		0	Normal	Unsupported
Low Operand ¹	0	+/- 0 ³	- High Operand	- High Operand
	Normal	Low Operand	Normal, +/- 0 ⁴	Undefined
	Unsupported ⁵	Low Operand	Undefined	Undefined

Note:

1. Least-significant floating-point value in first or second source operand.
2. Most-significant floating-point value in first or second source operand.
3. The sign is the logical AND of the sign of the low operand and the inverse of the sign of the high operand.
4. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the low operand is larger in magnitude than the high operand, the sign of this zero is the same as the sign of the low operand, else it is the inverse of the sign of the high operand. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of the low operand.
5. "Unsupported" means that the exponent is all ones (1s).

Table 2-26. Numeric Range of PFPNACC Result (High Result)

Source Operand		High Operand ²		
		0	Normal	Unsupported
Low Operand ¹	0	+/- 0 ³	High Operand	High Operand
	Normal	Low Operand	Normal, +/- 0 ⁴	Undefined
	Unsupported ⁵	Low Operand	Undefined	Undefined

Note:

1. Least-significant floating-point value in first or second source operand.
2. Most-significant floating-point value in first or second source operand.
3. The sign is the logical AND of the signs of the low and high operands.
4. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is zero with the sign of the operand (low or high) that is larger in magnitude. If the infinitely precise result is exactly zero, the result is zero with the sign of the low operand. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of the low operand.
5. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFADD, PFSUB, PFACC, PFPNACC

rFLAGS Affected

None.

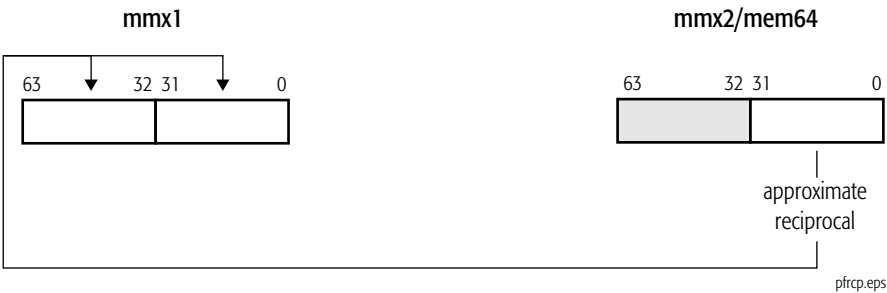
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD Extensions to 3DNow!™ bit (bit 30) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFRCP **Floating-Point Reciprocal Approximation**

The PFRCP instruction computes the approximate reciprocal of the single-precision floating-point value in the low-order 32 bits of an MMX register or 64-bit memory location and writes the result in both doublewords of another MMX register. The result is accurate to 14 bits.

Mnemonic	Opcode	Description
PFRCP <i>mmx1, mmx2/mem64</i>	0F 0F 96	Computes approximate reciprocal of single-precision floating-point value in an MMX register or 64-bit memory location and writes the result in both doublewords of the destination MMX register.



The PFRCP result can be forwarded to the Newton-Raphson iteration step 1 (PFRCPIT1) and Newton-Raphson iteration step 2 (PFRCPIT2) instructions to increase the accuracy of the reciprocal. The first stage of this refinement in accuracy (PFRCPIT1) requires that the input and output of the previously executed PFRCP instruction be used as input to the PFRCPIT1 instruction.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 2-27.

Table 2-27. Numeric Range for the PFRCP Result

Operand		Source 1 and Destination
Source 2	0	+/- Maximum Normal ¹
	Normal	Normal, +/- 0 ²
	Unsupported ³	Undefined

Note:

1. The result has the same sign as the source operand.
2. If the absolute value of the result is less than 2^{-126} , the result is zero with the sign being the sign of the source operand. Otherwise, the result is a normal with the sign being the same sign as the source operand.
3. "Unsupported" means that the exponent is all ones (1s).

Examples

The general Newton-Raphson recurrence for the reciprocal $1/b$ is:

$$Z_{i+1} \leftarrow Z_i \cdot (2 - b \cdot Z_i)$$

The following code sequence shows the computation of a/b :

```

X0 = PFRCP(b)
X1 = PFRCPIT1(b, X0)
X2 = PFRCPIT2(X1, X0)
q = PFMUL(a, X2)

```

The 24-bit final reciprocal value is X_2 . The quotient is formed in the last step by multiplying the reciprocal by the dividend a .

Related Instructions

PFRCPIT1, PFRCPIT2

rFLAGS Affected

None.

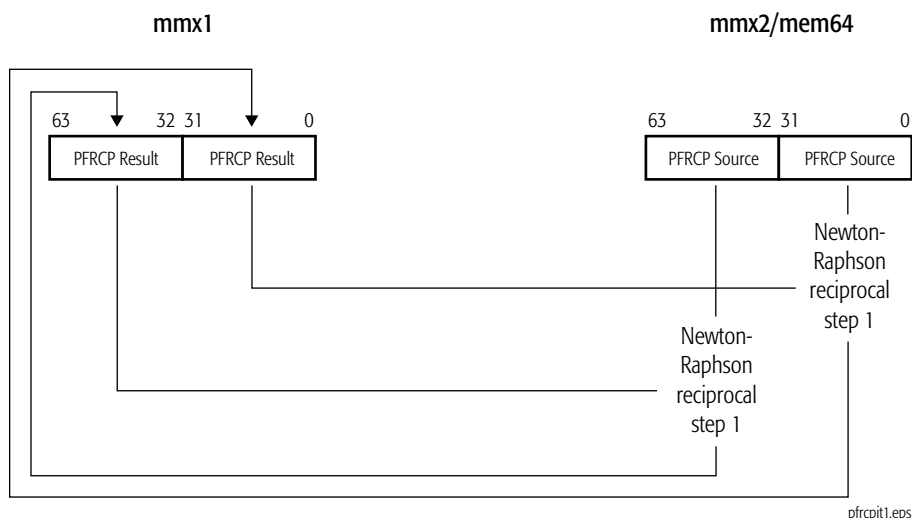
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFRCPIT1**Packed Floating-Point Reciprocal Iteration 1**

The PFRCPIT1 instruction performs the first step in the Newton-Raphson iteration to refine the reciprocal approximation produced by the PFRCP instruction. The first source/destination operand is an MMX register containing the results of two previous PFRCP instructions, and the second source operand is another MMX register or 64-bit memory location containing the source operands from the same PFRCP instructions.

Mnemonic	Opcode	Description
PFRCPIT1 <i>mmx1, mmx2/mem64</i>	0F 0F A6	Refine approximate reciprocal of result from previous PFRCP instruction.



This instruction is only defined for those combinations of operands such that the first source operand (*mmx1*) is the approximate reciprocal of the second source operand (*mmx2/mem64*), and thus the range of the product, $\text{mmx1} * \text{mmx2/mem64}$, is (0.5, 2). The initial approximation of an operand is accurate to about 12 bits, and the length of the operand itself is 24 bits, so the product of these two operands is greater than 24 bits. PFRCPIT1 applies the one's complement of the product and rounds the result to 32 bits. It then compresses the result to fit into 24 bits by removing the 8 redundant most-significant bits after the hidden integer bit.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or

other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 2-28.

Table 2-28. Numeric Range for the PFRCPIT1 Results

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	+/- 0 ¹	+/- 0 ¹
	Normal	+/- 0 ¹	Normal ²	Undefined
	Unsupported ³	+/- 0 ¹	Undefined	Undefined
Note: 1. The sign of the result is the exclusive-OR of the signs of the source operands. 2. The sign is positive. 3. "Unsupported" means that the exponent is all ones (1s).				

Operation

```
mmx1[31:0] = Compress (2 - mmx1[31:0] * (mmx2/mem64[31:0]) - 231);
mmx1[63:32] = Compress (2 - mmx1[63:32] * (mmx2/mem64[63:32]) - 231);
```

where:

“Compress” means discard the 8 redundant most-significant bits after the hidden integer bit.

Examples

The general Newton-Raphson recurrence for the reciprocal 1/b is:

$$Z_{i+1} \leftarrow Z_i \cdot (2 - b \cdot Z_i)$$

The following code sequence computes a 24-bit approximation to a/b with one Newton-Raphson iteration:

```
X0 = PFRCP(b)
X1 = PFRCPIT1(b, X0)
X2 = PFRCPIT2(X1, X0)
q = PFMUL(a, X2)
```

a/b is formed in the last step by multiplying the reciprocal approximation by a.

Related Instructions

PFRCP, PFRCPIT2

rFLAGS Affected

None.

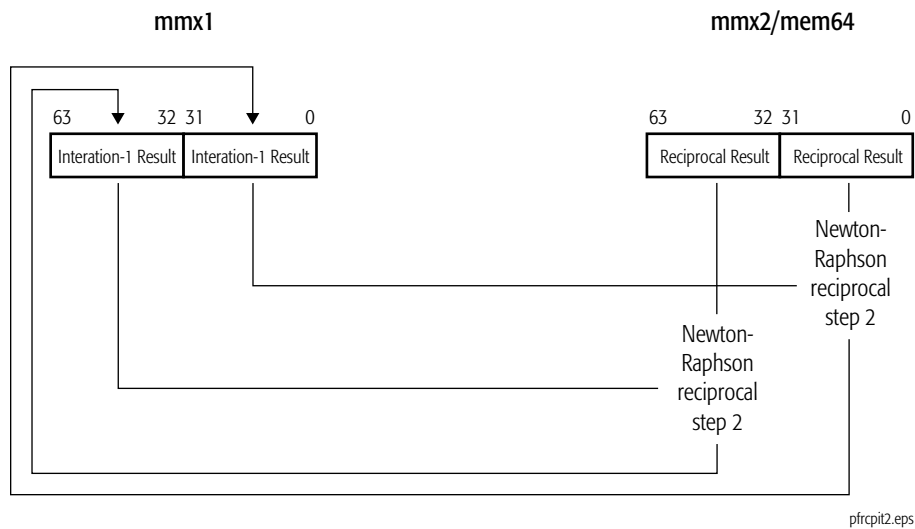
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFRCPIT2**Packed Floating-Point Reciprocal or Reciprocal Square Root Iteration 2**

The PFRCPIT2 instruction performs the second and final step in the Newton-Raphson iteration to refine the reciprocal approximation produced by the PFRCP instruction or the reciprocal square-root approximation produced by the PFSQRT instruction. PFRCPIT2 takes two paired elements in each source operand. These paired elements are the results of a PFRCP and PFRCPIT1 instruction sequence or of a PFRSQRT and PFRSQIT1 instruction sequence. The first source/destination operand is an MMX register that contains the PFRCPIT1 or PFRSQIT1 results and the second source operand is another MMX register or 64-bit memory location that contains the PFRCP or PFRSQRT results.

Mnemonic	Opcode	Description
PFRCPIT2 <i>mmx1, mmx2/mem64</i>	0F 0F B6	Refines approximate reciprocal result from previous PFRCP and PFRCPIT1 instructions or from previous PFRSQRT and PFRSQIT1 instructions.



The PFRCPIT2 instruction expands the compressed PFRCPIT1 or PFRSQIT1 results from 24 to 32 bits and multiplies them by their respective source operands. An optimal correction factor is added to the product, which is then rounded to 24 bits.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value

for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 2-29.

Table 2-29. Numeric Range for the PFRCPIT2 Results

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	+/- 0 ¹	+/- 0 ¹
	Normal	+/- 0 ¹	Normal, +/- 0 ²	Undefined
	Unsupported ³	+/- 0 ¹	Undefined	Undefined
Note: <ol style="list-style-type: none"> 1. The sign of the result is the exclusive-OR of the signs of the source operands. 2. If the absolute value of the result is less than 2^{-126}, the result is zero with the sign being the exclusive-OR of the signs of the source operands. If the absolute value of the product is greater than or equal to 2^{128}, the result is the largest normal number with the sign being exclusive-OR of the signs of the source operands. 3. "Unsupported" means that the exponent is all ones (1s). 				

Operation

```
mmx1[31:0] = Expand(mmx1[31:0]) * mmx2/mem64[31:0];
mmx1[63:32] = Expand(mmx1[63:32]) * mmx2/mem64[63:32];
```

where:

“Expand” means convert a 24-bit significand to a 32-bit significand according to the following rule:

```
temp[31:0] = {1'b1, 8{mmx1[22]}, mmx1[22:0]};
```

Examples

The general Newton-Raphson recurrence for the reciprocal 1/b is:

$$Z_{i+1} \leftarrow Z_i \cdot (2 - b \cdot Z_i)$$

The following code sequence computes a 24-bit approximation to a/b with one Newton-Raphson iteration:

```
X0 = PFRCP(b)
X1 = PFRCPIT1(b, X0)
X2 = PFRCPIT2(X1, X0)
q = PFMUL(a, X2)
```

a/b is formed in the last step by multiplying the reciprocal approximation by a.

Related Instructions

PFRCP, PFRCPIT1, PFRSQRT, PFRSQIT1

rFLAGS Affected

None.

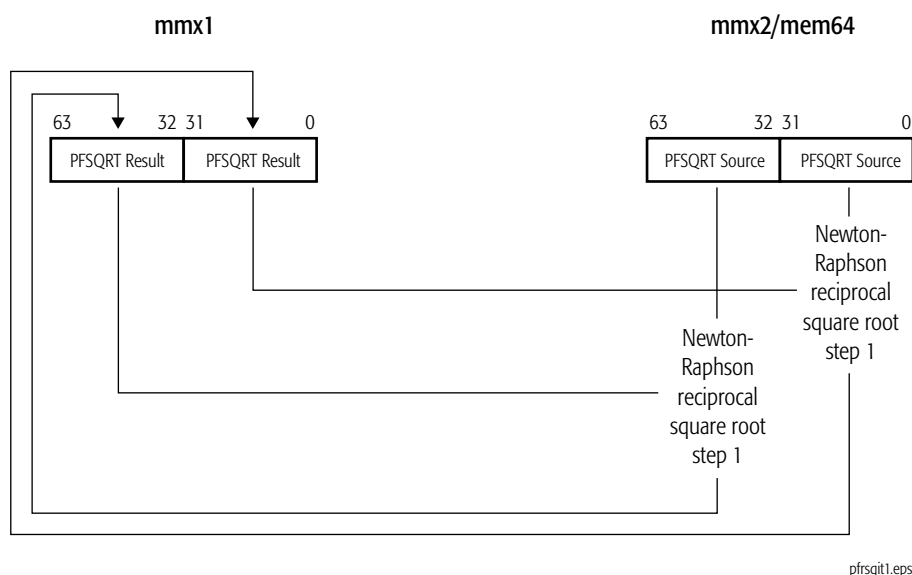
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFRSQIT1**Packed Floating-Point Reciprocal Square Root
Iteration 1**

The PFRSQIT1 instruction performs the first step in the Newton-Raphson iteration to refine the reciprocal square-root approximation produced by the PFSQRT instruction. The first source/destination operand is an MMX register containing the result from a previous PFRSQRT instruction, and the second source operand is another MMX register or 64-bit memory location containing the source operand from the same PFRSQRT instruction.

Mnemonic	Opcode	Description
PFRCPIT1 <i>mmx1, mmx2/mem64</i>	0F 0F A7	Refines reciprocal square root approximation of previous PFRSQRT instruction.



This instruction is only defined for those combinations of operands such that the first source operand (*mmx1*) is the approximate reciprocal of the second source operand (*mmx2/mem64*), and thus the range of the product, $\text{mmx1} * \text{mmx2/mem64}$, is (0.5, 2). The length of both operands is 24 bits, so the product of these two operands is greater than 24 bits. The product is normalized and then rounded to 32 bits. The one's complement of the result is applied, a 1 is added as the most-significant bit, and the result re-normalized. The result is then compressed to fit into 24 bits by removing 8 redundant most-significant bits after the hidden integer bit, and the exponent is reduced by 1 to account for the division by 2.

The numeric range for operands is shown in Table 2-30.

Table 2-30. Numeric Range for the PFRSQIT1 Result

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	+/- 0 ¹	+/- 0 ¹
	Normal	+/- 0 ¹	Normal ²	Undefined
	Unsupported ³	+/- 0 ¹	Undefined	Undefined

Note:

1. The sign of the result is the exclusive-OR of the signs of the source operands.
2. The sign is 0.
3. "Unsupported" means that the exponent is all ones (1s).

Operation

$$\begin{aligned} \text{mmx1}[31:0] &= \text{Compress}((3 - \text{mmx1}[31:0] * (\text{mmx2}/\text{mem64}[31:0]) - 2^{31})/2); \\ \text{mmx1}[63:32] &= \text{Compress}((3 - \text{mmx1}[63:32] * (\text{mmx2}/\text{mem64}[63:32]) - 2^{31})/2); \end{aligned}$$

where:

“Compress” means discard the 8 redundant most-significant bits after the hidden integer bit.

Examples

The following code sequence shows how the PFRSQRT and PFMUL instructions can be used to compute $a = 1/\text{sqrt}(b)$:

```
X0 = PFRSQRT(b)
X1 = PFMUL(X0,X0)
X2 = PFRSQIT1(b,X1)
a = PFRCPIT2(X2,X0)
```

Related Instructions

PFRCPIT2, PFRSQRT

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFRSQRT**Packed Floating-Point Reciprocal Square Root Approximation**

The PFRSQRT instruction computes the approximate reciprocal square root of the single-precision floating-point value in the low-order 32 bits of an MMX register or 64-bit memory location and writes the result in each doubleword of another MMX register. The source operand is single-precision with a 24-bit significand, and the result is accurate to 15 bits. Negative operands are treated as positive operands for purposes of reciprocal square-root computation, with the sign of the result the same as the sign of the source operand.

Mnemonic	Opcode	Description
PFRSQRT <i>mmx1, mmx2/mem64</i>	0F 0F 97	Computes approximate reciprocal square root of a packed single-precision floating-point value.



This instruction can be used together with the PFRSQIT1 and PFRCPIT2 instructions to increase accuracy. The first stage of this refinement in accuracy (PFRSQIT1) requires that the input and output of the previously executed PFRSQRT instruction be used as input to the PFRSQIT1 instruction.

The estimate contains the correct round-to-nearest value for approximately 99% of all arguments. The remaining arguments differ from the correct round-to-nearest value for the reciprocal by 1 unit-in-the-last-place (ulp). For details, see the data sheet or other software-optimization documentation relating to particular hardware implementations.

The numeric range for operands is shown in Table 2-31.

Table 2-31. Numeric Range for the PFRCP Result

Operand		Source 1 and Destination
Source 2	0	+/- Maximum Normal ¹
	Normal	Normal ¹
	Unsupported ²	Undefined ¹
Note: 1. The result has the same sign as the source operand. 2. "Unsupported" means that the exponent is all ones (1s).		

Examples

The following code sequence shows how the PFRSQRT and PFMUL instructions can be used to compute $a = 1/\sqrt{b}$:

```

X0 = PFRSQRT(b)
X1 = PFMUL(X0, X0)
X2 = PFRSQIT1(b, X1)
a = PFRCPIT2(X2, X0)

```

Related Instructions

PFRCPIT2, PFRSQIT1

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFSUB**Packed Floating-Point Subtract**

The PFSUB instruction subtracts each packed single-precision floating-point value in the second source operand from the corresponding packed single-precision floating-point value in the first source operand and writes the result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-32.

Mnemonic	Opcode	Description
PFSUB <i>mmx1</i> , <i>mmx2/mem64</i>	0F 0F 9A	Subtracts packed single-precision floating-point values in an MMX register or 64-bit memory location from packed single-precision floating-point values in another MMX register and writes the result in the destination MMX register.

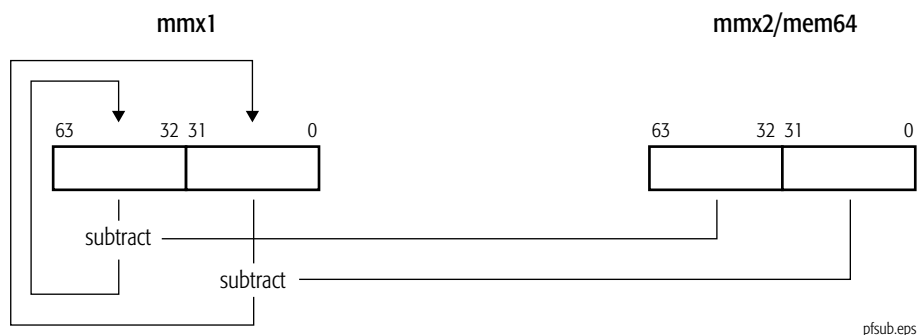


Table 2-32. Numeric Range for the PFSUB Results

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	- Source 2	- Source 2
	Normal	Source 1	Normal, +/- 0 ²	Undefined
	Unsupported ³	Source 1	Undefined	Undefined
Note: <ol style="list-style-type: none"> The sign of the result is the logical AND of the sign of source 1 and the inverse of the sign of source 2 If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the source operand that is larger in magnitude is source 1, the sign of this zero is the same as the sign of source 1, else it is the inverse of the sign of source 2. If the infinitely precise result is exactly zero, the result is zero with the sign of source 1. If the absolute value of the infinitely precise result is greater than or equal to 2^{128}, the result is the largest normal number with the sign of source 1. "Unsupported" means that the exponent is all ones (1s). 				

Related Instructions

PFSUBR

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PFSUBR**Packed Floating-Point Subtract Reverse**

The PFSUBR instruction subtracts each packed single-precision floating-point value in the first source operand from the corresponding packed single-precision floating-point value in the second source operand and writes the result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location. The numeric range for operands is shown in Table 2-33.

Mnemonic	Opcode	Description
PFSUBR <i>mmx1, mmx2/mem64</i>	0F 0F AA	Subtracts packed single-precision floating-point values in an MMX register from packed single-precision floating-point values in another MMX register or 64-bit memory location and writes the result in the destination MMX register.

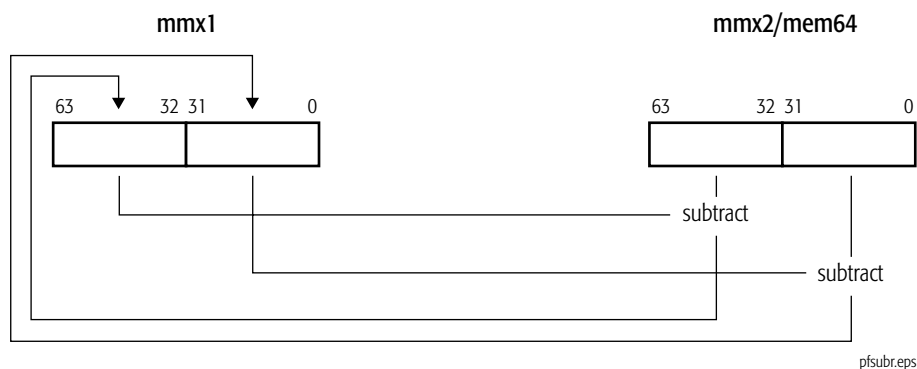


Table 2-33. Numeric Range for the PFSUBR Results

Source Operand		Source 2		
		0	Normal	Unsupported
Source 1 and Destination	0	+/- 0 ¹	Source 2	Source 2
	Normal	- Source 1	Normal, +/- 0 ²	Undefined
	Unsupported ³	- Source 1	Undefined	Undefined

Note:

1. The sign is the logical AND of the sign of source 2 and the inverse of the sign of source 1.
2. If the absolute value of the infinitely precise result is less than 2^{-126} (but not zero), the result is a zero. If the source operand that is larger in magnitude is source 2, the sign of this zero is the same as the sign of source 2, else it is the inverse of the sign of source 1. If the infinitely precise result is exactly zero, the result is zero with the sign of source 2. If the absolute value of the infinitely precise result is greater than or equal to 2^{128} , the result is the largest normal number with the sign of source 2.
3. "Unsupported" means that the exponent is all ones (1s).

Related Instructions

PFSUB

rFLAGS Affected

None.

Exceptions0

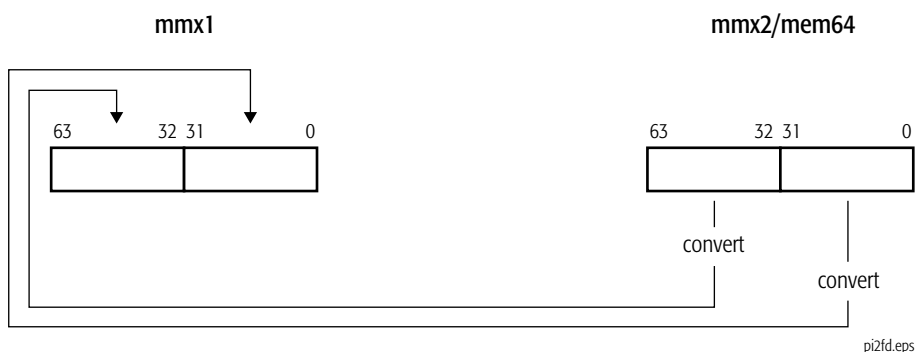
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PI2FD**Packed Integer to Floating-Point Doubleword Conversion**

The PI2FD instruction converts two packed 32-bit signed integer values in an MMX register or a 64-bit memory location to two packed single-precision floating-point values and writes the converted values in another MMX register. If the result of the conversion is an inexact value, the value is truncated (rounded toward zero).

Mnemonic	Opcode	Description
PI2FD <i>mmx1, mmx2/mem64</i>	0F 0F 0D	Converts packed doubleword integers in an MMX register or 64-bit memory location to single-precision floating-point values in the destination MMX register. Inexact results are truncated.

**Related Instructions**

PF2ID, PF2IW, PI2FW

rFLAGS Affected

None

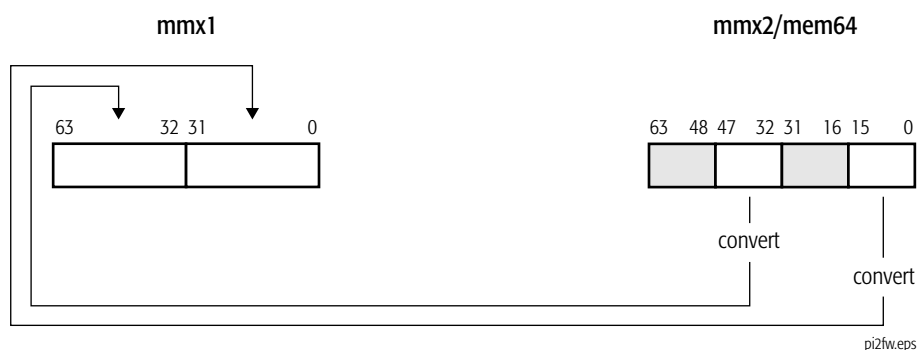
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PI2FW**Packed Integer to Floating-Point Word Conversion**

The PI2FW instruction converts two packed 16-bit signed integer values in an MMX register or a 64-bit memory location to two packed single-precision floating-point values and writes the converted values in another MMX register.

Mnemonic	Opcode	Description
PI2FW <i>mmx1, mmx2/mem64</i>	0F 0F 0C	Converts packed 16-bit integers in an XMM register or 64-bit memory location to packed single-precision floating-point values in the destination MMX register.

**Related Instructions**

PF2ID, PF2IW, PI2FD

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD Extensions to 3DNow!™ bit (bit 30) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PINSRW Packed Insert Word

The PINSRW instruction inserts a 16-bit value from the low-order word of a 32-bit or 64-bit general purpose register or a 16-bit memory location into an MMX register. The location in the destination register is selected by the immediate byte operand, as shown in Table 2-34. The other words in the destination register operand are not modified.

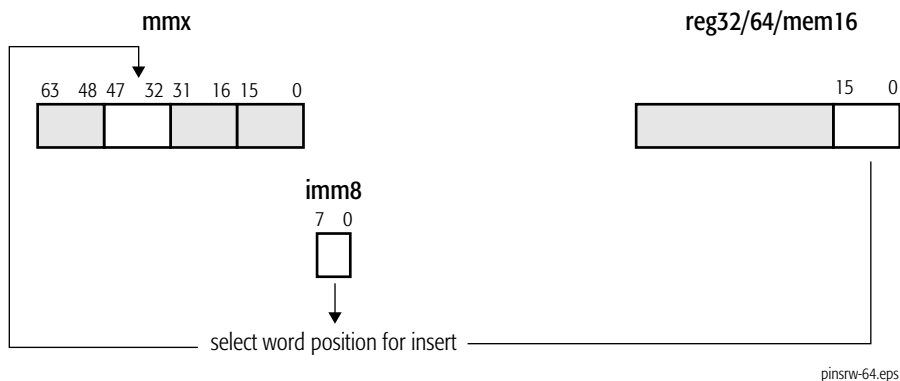
Mnemonic

PINSRW *mmx, reg32/64/mem16, imm8* OF C4/r ib

Opcode

Description

Inserts a 16-bit value from a general-purpose register or memory location into an MMX register.



pinsrw-64.eps

Table 2-34. Immediate-Byte Operand Encoding for 64-Bit PINSRW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled
1:0	0	15:0
	1	31:16
	2	47:32
	3	63:48

Related Instructions

PEXTRW

rFLAGS Affected

None.

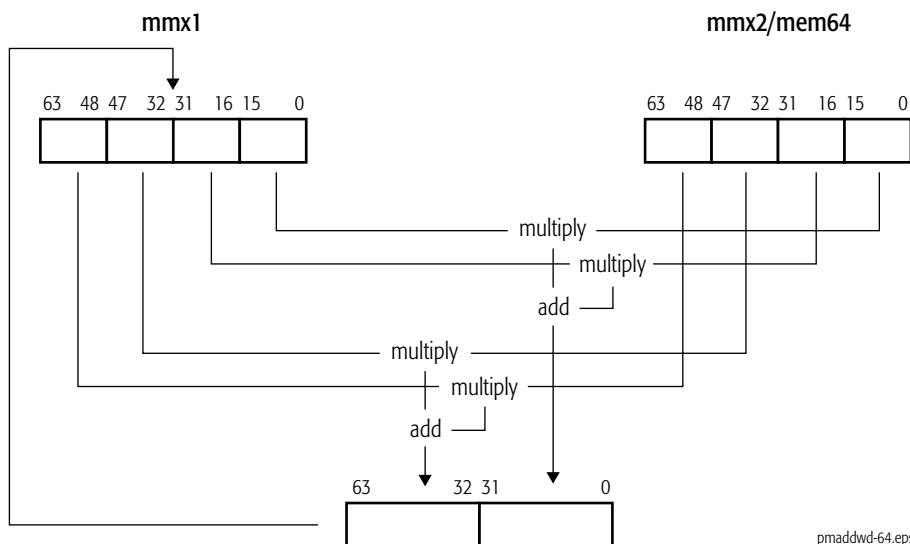
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMADDWD**Packed Multiply Words and Add Doublewords**

The PMADDWD instruction multiplies each packed 16-bit signed value in the first source operand by the corresponding packed 16-bit signed value in the second source operand, adds the adjacent intermediate 32-bit results of each multiplication (for example, the multiplication results for the adjacent bit fields 63:48 and 47:32, and 31:16 and 15:0), and writes the 32-bit result of each addition in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMADDWD <i>mmx1, mmx2/mem64</i>	0F F5/r	Multiplies four packed 16-bit signed values in an MMX register and another MMX register or 64-bit memory location, adds intermediate results, and writes the result in the destination MMX register.



If all four of the 16-bit source operands used to produce a 32-bit multiply-add result have the value 8000h, the 32-bit result is 8000_0000h, which is incorrect.

Related Instructions

PMULHUW, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None.

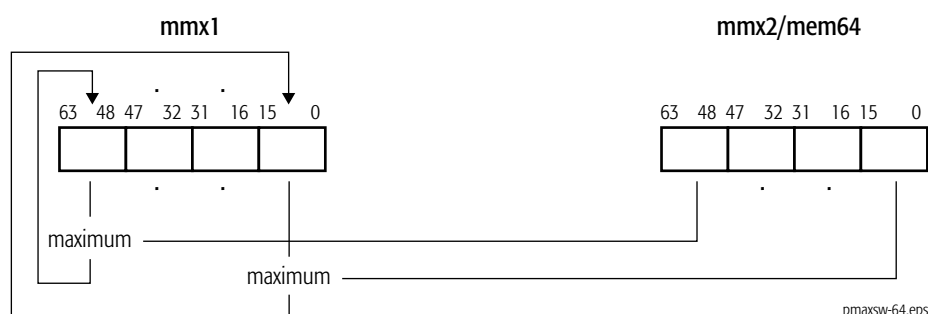
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMAXSW**Packed Maximum Signed Words**

The PMAXSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMAXSW <i>mmx1, mmx2/mem64</i>	0F EE /r	Compares packed signed 16-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the maximum value of each compare in destination MMX register.

**Related Instructions**

PMAXUB, PMINSW, PMINUB

rFLAGS Affected

None.

Exceptions

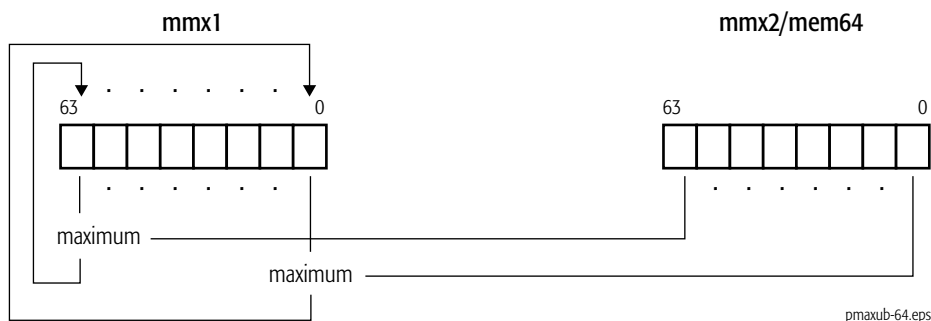
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMAXUB

Packed Maximum Unsigned Bytes

The PMAXUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the maximum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMAXUB <i>mmx1, mmx2/mem64</i>	0F DE /r	Compares packed unsigned 8-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the maximum value of each compare in the destination MMX register.



Related Instructions

PMAXSW, PMINSW, PMINUB

rFLAGS Affected

None.

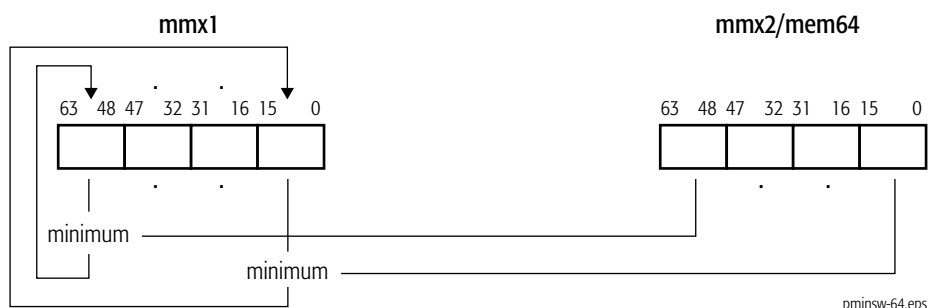
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMINSW**Packed Minimum Signed Words**

The PMINSW instruction compares each of the packed 16-bit signed integer values in the first source operand with the corresponding packed 16-bit signed integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding word of the destination (first source). The first source/destination and second source operands are an MMX register and an MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMINSW <i>mmx1, mmx2/mem64</i>	0F EA/r	Compares packed signed 16-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the minimum value of each compare in the destination MMX register.

**Related Instructions**

PMAXSW, PMAXUB, PMINUB

rFLAGS Affected

None.

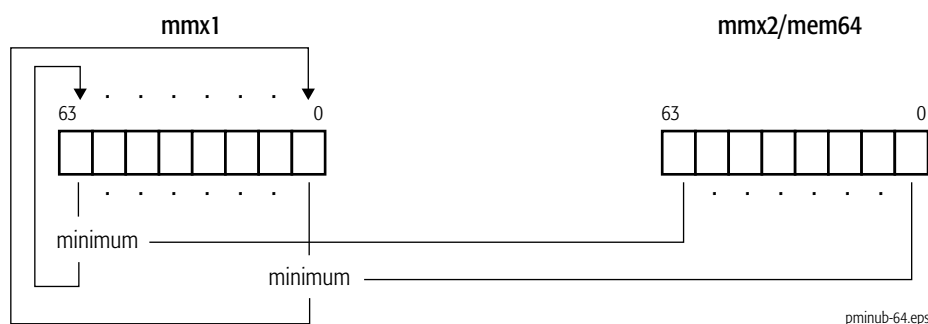
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMINUB**Packed Minimum Unsigned Bytes**

The PMINUB instruction compares each of the packed 8-bit unsigned integer values in the first source operand with the corresponding packed 8-bit unsigned integer value in the second source operand and writes the minimum of the two values for each comparison in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMINUB <i>mmx1, mmx2/mem64</i>	0F DA/ <i>r</i>	Compares packed unsigned 8-bit integer values in an MMX register and another MMX register or 64-bit memory location and writes the minimum value of each comparison in the destination MMX register.

**Related Instructions**

PMAWS, PMAUB, PMINSW

rFLAGS Affected

None.

Exceptions

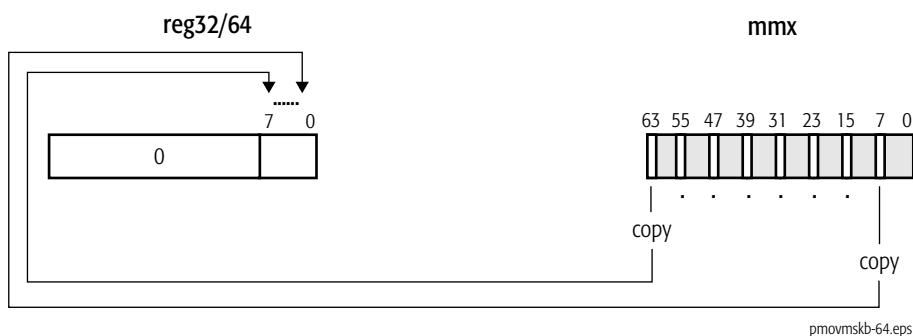
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMOVMSKB**Packed Move Mask Byte**

The PMOVMSKB instruction moves the most-significant bit of each byte in the source operand to the destination, with zero-extension to 32 or 64 bits. The destination and source operands are a 32-bit or 64-bit general-purpose register and an MMX register.

If the source operand is an XMM register, the result is written to the low-order word of the general-purpose register. If the source operand is an MMX register, the result is written to the low-order byte of the general-purpose register.

Mnemonic	Opcode	Description
PMOVMSKB <i>reg32/64, mmx</i>	0F D7 /r	Moves most-significant bit of each byte in an MMX register to the low-order byte of a 32-bit or 64-bit general-purpose register.

**Related Instructions**

MOVMSKPD, MOVMSKPS

rFLAGS Affected

None.

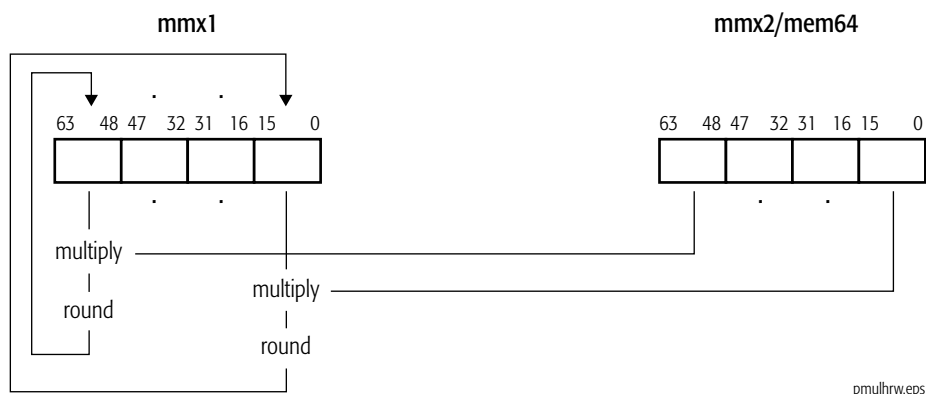
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

PMULHRW**Packed Multiply High Rounded Word**

The PMULHRW instruction multiplies each of the four packed 16-bit signed integer values in the first source operand by the corresponding packed 16-bit integer value in the second source operand, adds 8000h to the lower 16 bits of the intermediate 32-bit result of each multiplication, and writes the high-order 16 bits of each result in the corresponding word of the destination (first source). The addition of 8000h results in the rounding of the result, providing a numerically more accurate result than the PMULHW instruction, which truncates the result. The first source/destination operand is an MMX register. The second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULHRW <i>mmx1, mmx2/mem64</i>	0F 0F B7	Multiply 16-bit signed integer values in an MMX register and another MMX register or 64-bit memory location and write rounded result in the destination MMX register.



pmulhrw.eps

Related Instructions

None.

rFLAGS Affected

None.

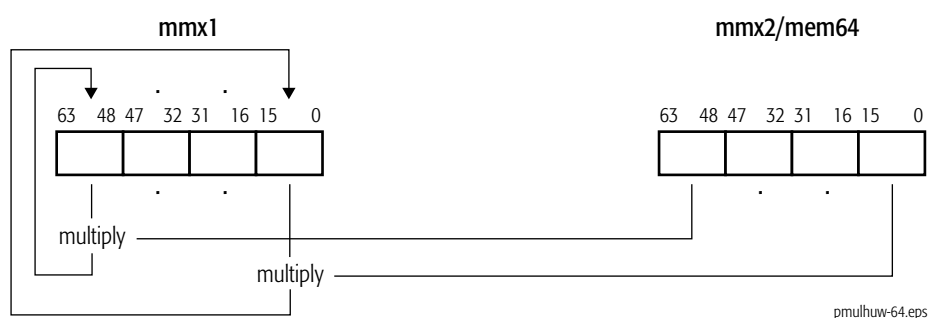
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD 3DNow!™ bit (bit 31) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMULHUW**Packed Multiply High Unsigned Word**

The PMULHUW instruction multiplies each packed unsigned 16-bit values in the first source operand by the corresponding packed unsigned word in the second source operand and writes the high-order 16 bits of each intermediate 32-bit result in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULHUW <i>mmx1, mmx2/mem64</i>	0F E4 /r	Multiplies packed 16-bit values in an MMX register by the packed 16-bit values in another MMX register or 64-bit memory location and writes the high-order 16 bits of each result in the destination MMX register.

**Related Instructions**

PMADDWD, PMULHW, PMULLW, PMULUDQ

rFLAGS Affected

None.

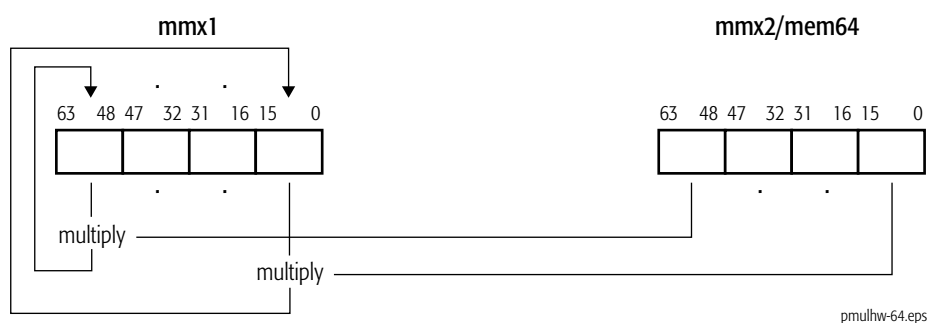
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMULHW**Packed Multiply High Signed Word**

The PMULHW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the high-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULHW <i>mmx1, mmx2/mem64</i>	0F E5 /r	Multiplies packed 16-bit signed integer values in an MMX register and another MMX register or 64-bit memory location and writes the high-order 16 bits of each result in the destination MMX register.

**Related Instructions**

PMADDWD, PMULHUW, PMULLW, PMULUDQ

rFLAGS Affected

None.

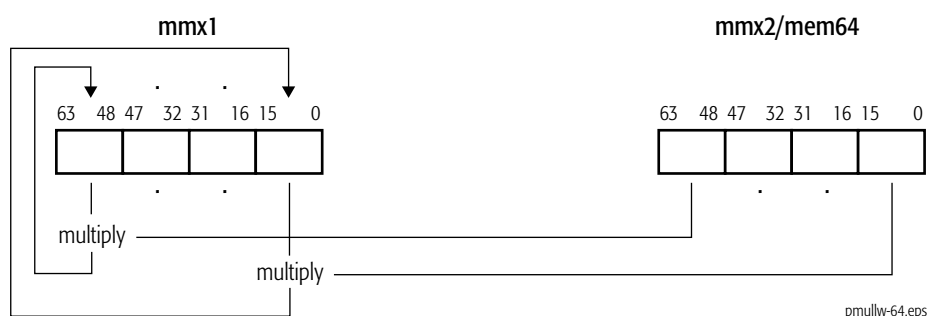
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMULLW**Packed Multiply Low Signed Word**

The PMULLW instruction multiplies each packed 16-bit signed integer value in the first source operand by the corresponding packed 16-bit signed integer in the second source operand and writes the low-order 16 bits of the intermediate 32-bit result of each multiplication in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULLW <i>mmx1, mmx2/mem64</i>	0F D5 /r	Multiplies packed 16-bit signed integer values in an MMX register and another MMX register or 64-bit memory location and writes the low-order 16 bits of each result in the destination MMX register.

**Related Instructions**

PMADDWD, PMULHUW, PMULHW, PMULUDQ

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PMULUDQ**Packed Multiply Unsigned Doubleword and Store Quadword**

The PMULUDQ instruction multiplies two 32-bit unsigned integer values in the low-order doubleword of the first and second source operands and writes the 64-bit result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PMULUDQ <i>mmx1, mmx2/mem64</i>	0F F4 /r	Multiplies low-order 32-bit unsigned integer value in an MMX register and another MMX register or 64-bit memory location and writes the 64-bit result in the destination MMX register.

**Related Instructions**

PMADDWD, PMULHUW, PMULHW, PMULLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

POR Packed Logical Bitwise OR

The POR instruction performs a bitwise logical OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
POR <i>mmx1, mmx2/mem64</i>	OF EB/r	Performs bitwise logical OR of values in an MMX register and in another MMX register or 64-bit memory location and writes the result in the destination MMX register.



por-64.eps

Related Instructions

PAND, PANDN, PXOR

rFLAGS Affected

None.

Exceptions

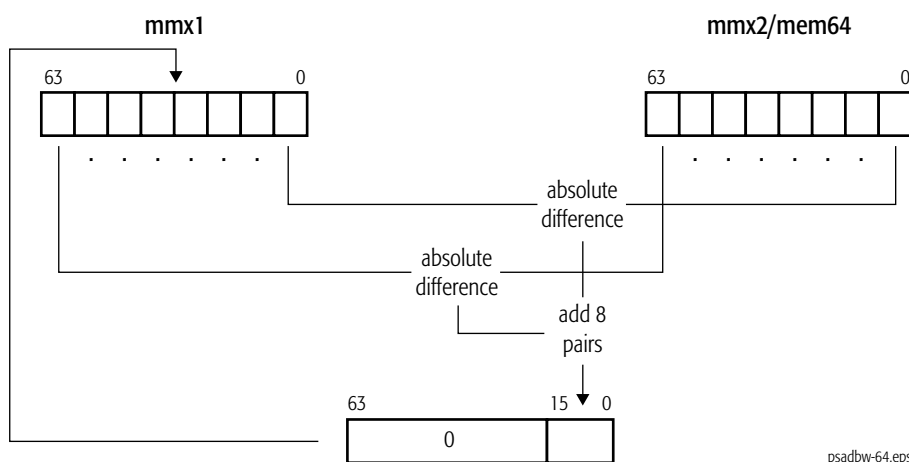
Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSADBW**Packed Sum of Absolute Differences of Bytes Into a Word**

The PSADBW instruction computes the absolute differences of eight corresponding packed 8-bit unsigned integers in the first and second source operands and writes the unsigned 16-bit integer result of the sum of the eight differences in a word in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location. The result is stored in the low-order word of the destination operand, and the remaining bytes in the destination are cleared to all 0s.

Mnemonic	Opcode	Description
PSADBW <i>mmx1, mmx2/mem64</i>	0F F6/ <i>r</i>	Compute the sum of the absolute differences of packed 8-bit unsigned integer values in an MMX register and another MMX register or 64-bit memory location and writes the 16-bit unsigned integer result in the destination MMX register.



psadbw-64.eps

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSHUFW**Packed Shuffle Words**

The PSHUFW instruction moves any one of the four packed words in an MMX register or 64-bit memory location to a specified word location in another MMX register. In each case, the selection of the value of the destination word is determined by a two-bit field in the immediate-byte operand, with bits 0 and 1 selecting the contents of the low-order word, bits 2 and 3 selecting the second word, bits 4 and 5 selecting the third word, and bits 6 and 7 selecting the high-order word. Refer to Table 2-35. A word in the source operand may be copied to more than one word in the destination.

Mnemonic	Opcode	Description
PSHUFW <i>mmx1, mmx2/mem64, imm8</i>	0F 70 /r <i>ib</i>	Shuffles packed 16-bit values in an MMX register or 64-bit memory location and puts the result in another XMM register.

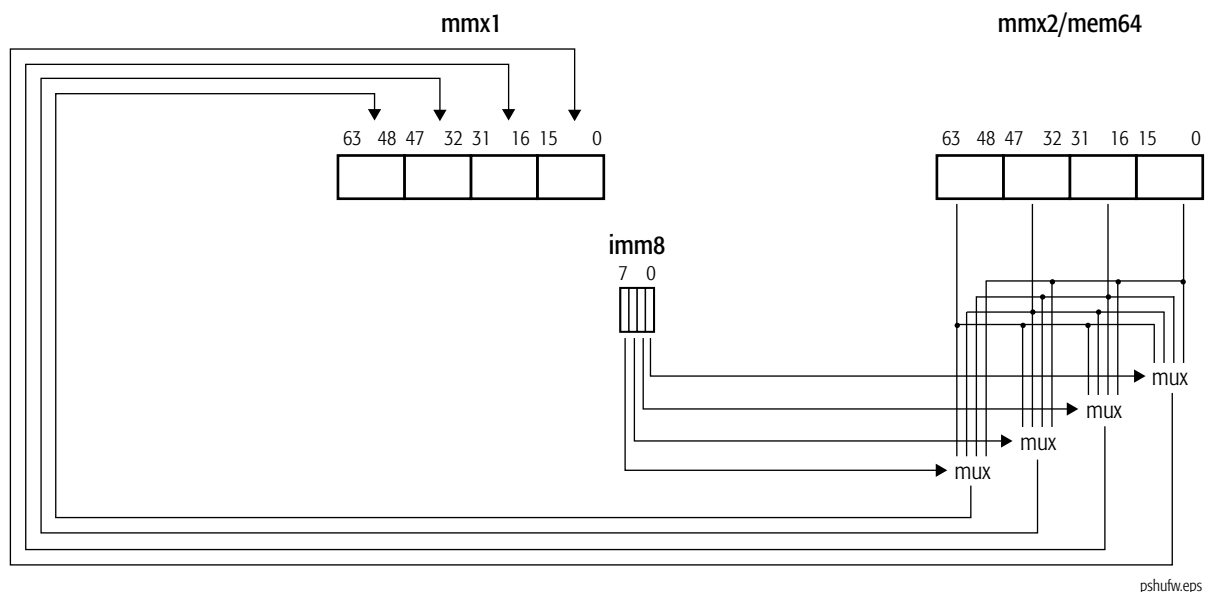


Table 2-35. Immediate-Byte Operand Encoding for PSHUFW

Immediate-Byte Bit Field	Value of Bit Field	Destination Bits Filled	Source Bits Moved
1:0	0	15:0	15:0
	1	15:0	31:16
	2	15:0	47:32
	3	15:0	63:48
3:2	0	31:16	15:0
	1	31:16	31:16
	2	31:16	47:32
	3	31:16	63:48
5:4	0	47:32	15:0
	1	47:32	31:16
	2	47:32	47:32
	3	47:32	63:48
7:6	0	63:48	15:0
	1	63:48	31:16
	2	63:48	47:32
	3	63:48	63:48

Related Instructions

PSHUFD, PSHUFW, PSHUFLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE bit (bit 25) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

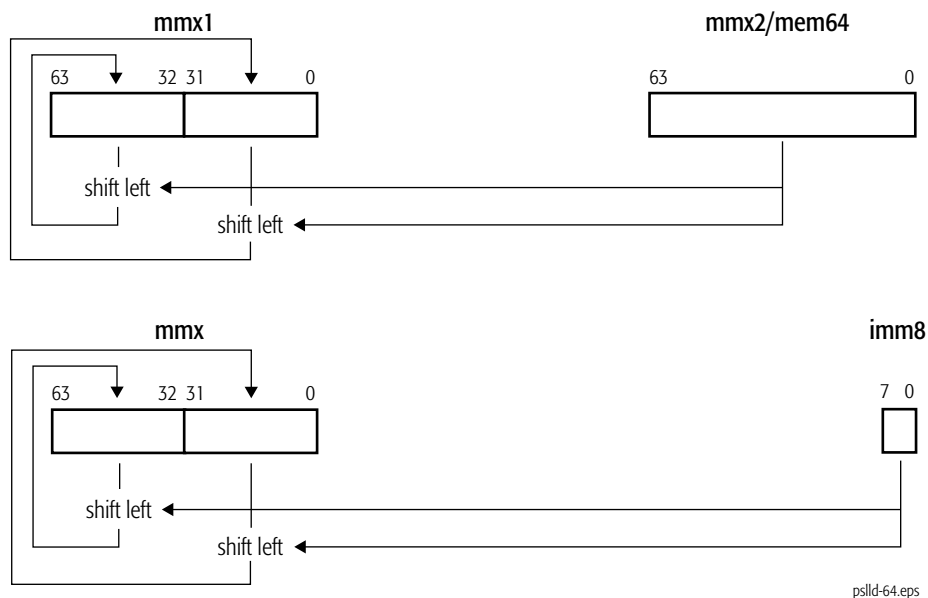
PSLLD**Packed Shift Left Logical Doublewords**

The PSLLD instruction left-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLD <i>mmx1, mmx2/mem64</i>	0F F2 /r	Left-shifts packed doublewords in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSLLD <i>mmx, imm8</i>	0F 72 /6 <i>ib</i>	Left-shifts packed doublewords in an MMX register by the amount specified in an immediate byte value.



pslld-64.eps

Related Instructions

PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

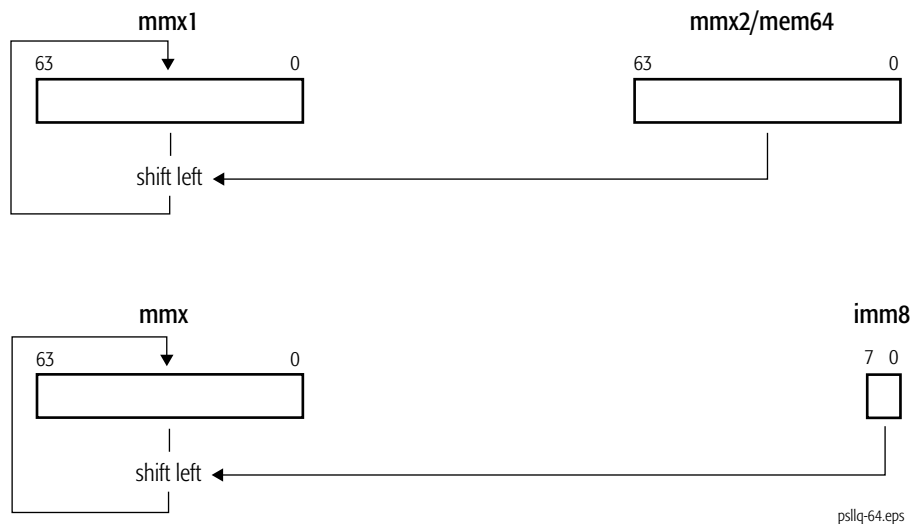
PSLLQ**Packed Shift Left Logical Quadwords**

The PSLLQ instruction left-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLQ <i>mmx1, mmx2/mem64</i>	0F F3 /r	Left-shifts quadword in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSLLQ <i>mmx imm8</i>	0F 73 /6 ib	Left-shifts quadword in an MMX register by the amount specified in an immediate byte value.

**Related Instructions**

PSLLD, PSLLDQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

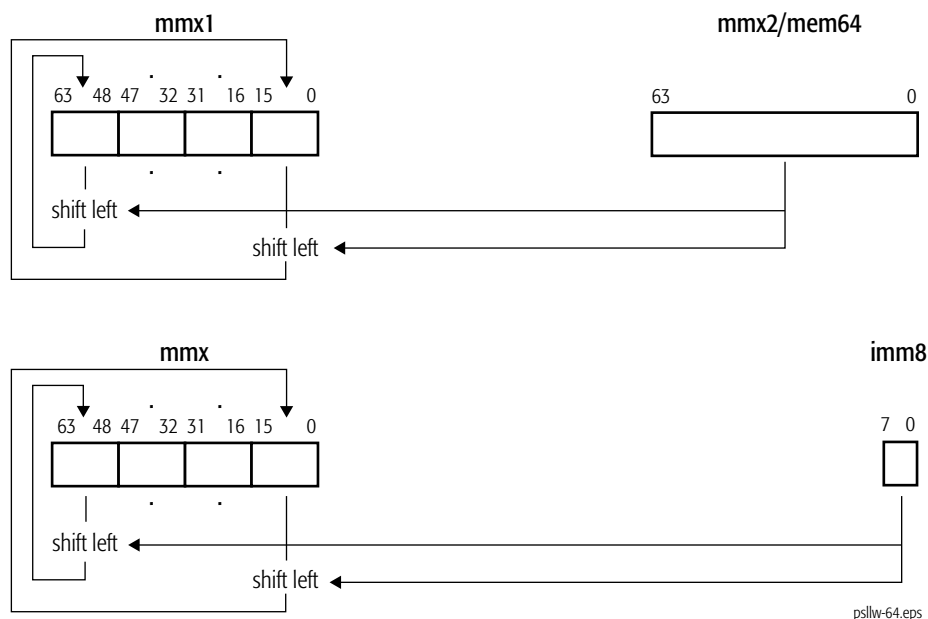
PSLLW**Packed Shift Left Logical Words**

The PSLLW instruction left-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The low-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to all 0s.

Mnemonic	Opcode	Description
PSLLW <i>mmx1, mmx2/mem64</i>	0F F1 /r	Left-shifts packed words in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSLLW <i>mmx, imm8</i>	0F 71 /6 ib	Left-shifts packed words in an MMX register by the amount specified in an immediate byte value.

**Related Instructions**

PSLLD, PSLLDQ, PSLLQ, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

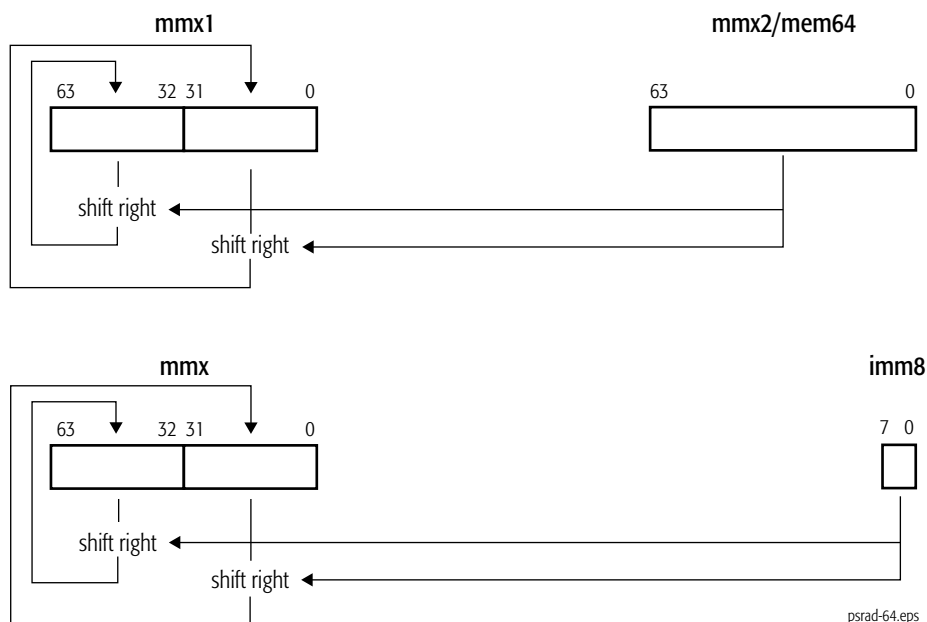
PSRAD**Packed Shift Right Arithmetic Doublewords**

The PSRAD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the doubleword's initial value. If the shift value is greater than 31, each doubleword in the destination is filled with the sign bit of the doubleword's initial value.

Mnemonic	Opcode	Description
PSRAD <i>mmx1, mmx2/mem64</i>	0F E2 /r	Right-shifts packed doublewords in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSRAD <i>mmx, imm8</i>	0F 72 /4 ib	Right-shifts packed doublewords in an MMX register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAW, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

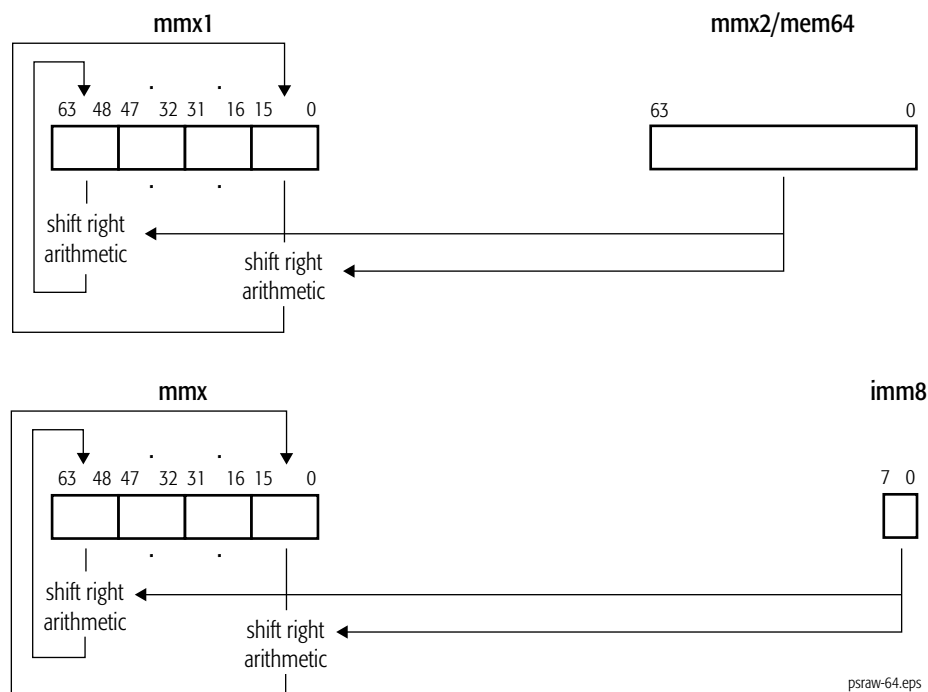
PSRAW**Packed Shift Right Arithmetic Words**

The PSRAW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are filled with the sign bit of the word's initial value. If the shift value is greater than 15, each word in the destination is filled with the sign bit of the word's initial value.

Mnemonic	Opcode	Description
PSRAW <i>mmx1, mmx2/mem64</i>	0F E1 /r	Right-shifts packed words in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSRAW <i>mmx, imm8</i>	0F 71 /4 ib	Right-shifts packed words in an MMX register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRLD, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

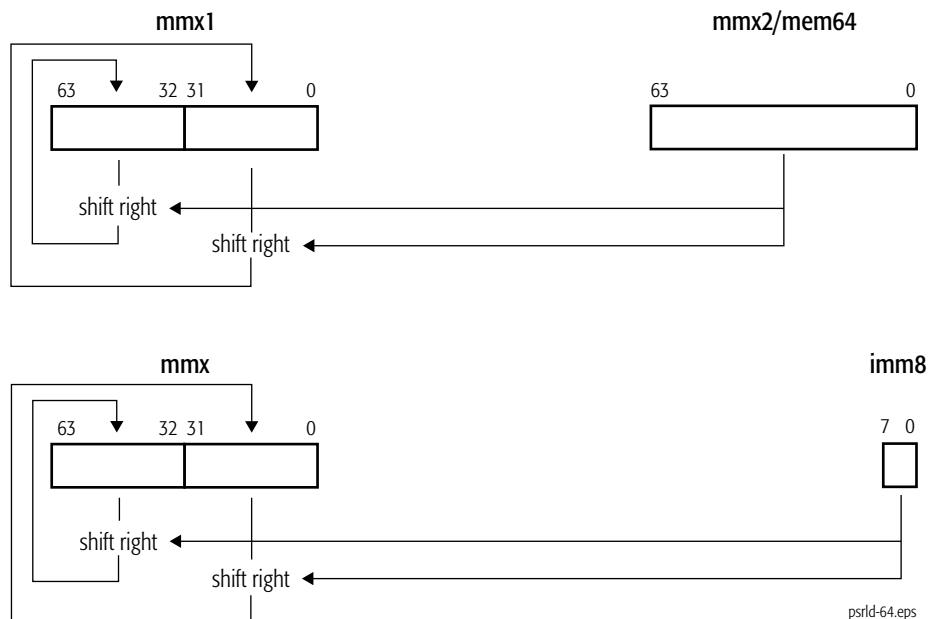
PSRLD**Packed Shift Right Logical Doublewords**

The PSRLD instruction right-shifts each of the packed 32-bit values in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding doubleword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 31, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLD <i>mmx1, mmx2/mem64</i>	0F D2 /r	Right-shifts packed doublewords in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSRLD <i>mmx, imm8</i>	0F 72 /2 <i>ib</i>	Right-shifts packed doublewords in an MMX register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLDQ, PSRLQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

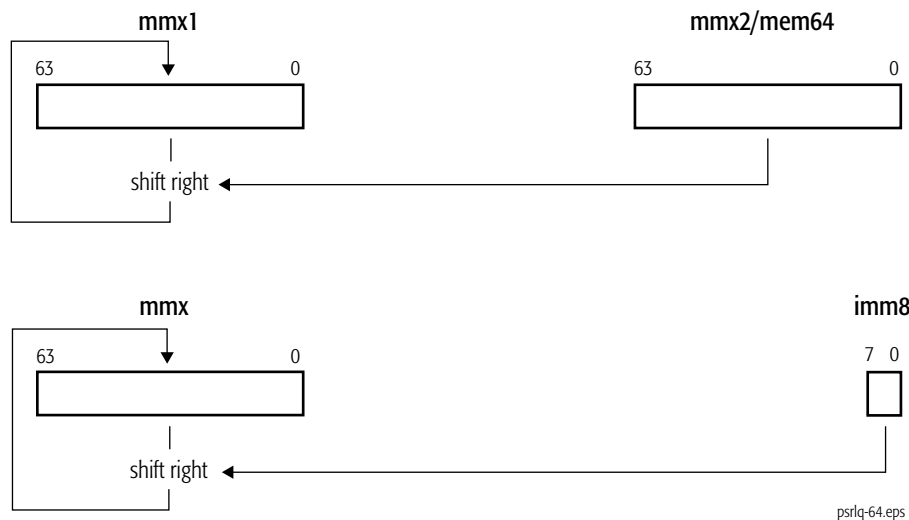
PSRLQ**Packed Shift Right Logical Quadwords**

The PSRLQ instruction right-shifts each 64-bit value in the first source operand by the number of bits specified in the second source operand and writes each shifted value in the corresponding quadword of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 63, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLQ <i>mmx1, mmx2/mem64</i>	0F D3 /r	Right-shifts quadword in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSRLQ <i>mmx, imm8</i>	0F 73 /2 ib	Right-shifts quadword in an MMX register by the amount specified in an immediate byte value.

**Related Instructions**

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

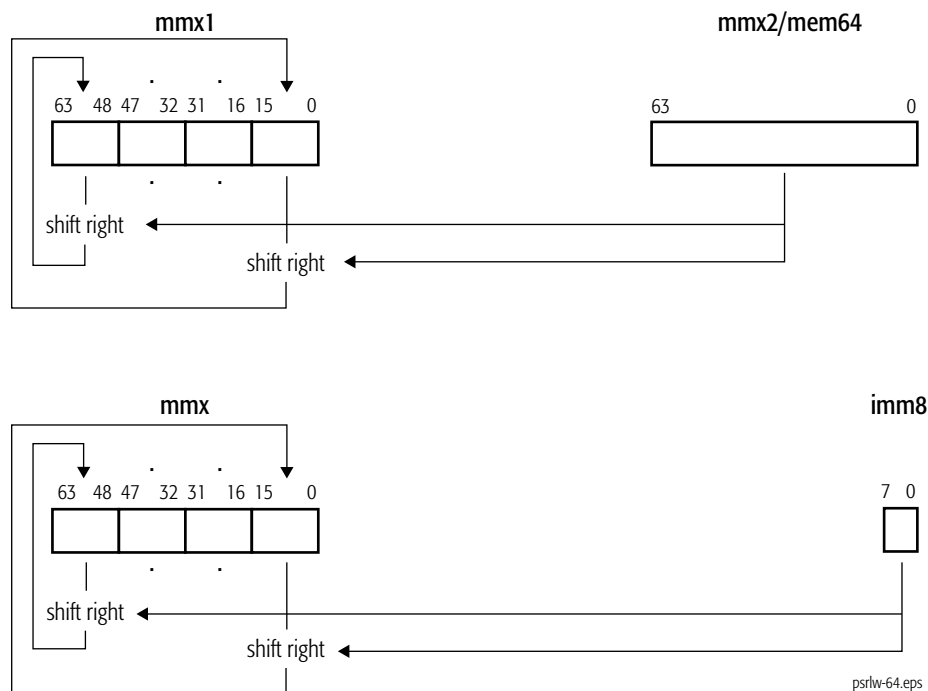
PSRLW**Packed Shift Right Logical Words**

The PSRLW instruction right-shifts each of the packed 16-bit values in the first source operand by the number of bits specified in the second operand and writes each shifted value in the corresponding word of the destination (first source). The first source/destination and second source operands are:

- an MMX register and another MMX register or 64-bit memory location, or
- an MMX register and an immediate byte value.

The high-order bits that are emptied by the shift operation are cleared to 0. If the shift value is greater than 15, the destination is cleared to 0.

Mnemonic	Opcode	Description
PSRLW <i>mmx1, mmx2/mem64</i>	0F D1 /r	Right-shifts packed words in an MMX register by the amount specified in an MMX register or 64-bit memory location.
PSRLW <i>mmx, imm8</i>	0F 71 /2 ib	Right-shifts packed words in an MMX register by the amount specified in an immediate byte value.



Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLDQ, PSRLQ

rFLAGS Affected

None.

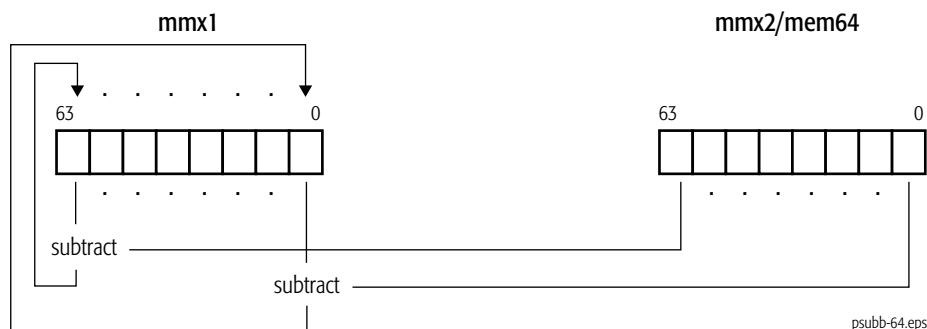
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBB**Packed Subtract Bytes**

The PSUBB instruction subtracts each packed 8-bit integer value in the second source operand from the corresponding packed 8-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBB <i>mmx1, mmx2/mem64</i>	0F F8 /r	Subtracts packed byte integer values in an MMX register or 64-bit memory location from packed byte integer values in another MMX register and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written in the destination.

Related Instructions

PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

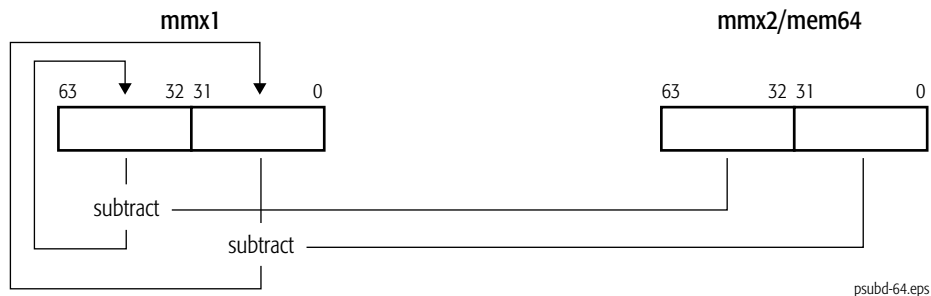
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBD**Packed Subtract Doublewords**

The PSUBD instruction subtracts each packed 32-bit integer value in the second source operand from the corresponding packed 32-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding doubleword of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBD <i>mmx1, mmx2/mem64</i>	0F FA/r	Subtracts packed 32-bit integer values in an MMX register or 64-bit memory location from packed 32-bit integer values in another MMX register and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBQ**Packed Subtract Quadword**

The PSUBQ instruction subtracts each packed 64-bit integer value in the second source operand from the corresponding packed 64-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding quadword of the destination (first source). The first source/destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBQ <i>mmx1, mmx2/mem64</i>	0F FB /r	Subtracts packed 64-bit integer values in an MMX register or 64-bit memory location from packed 64-bit integer values in another MMX register and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

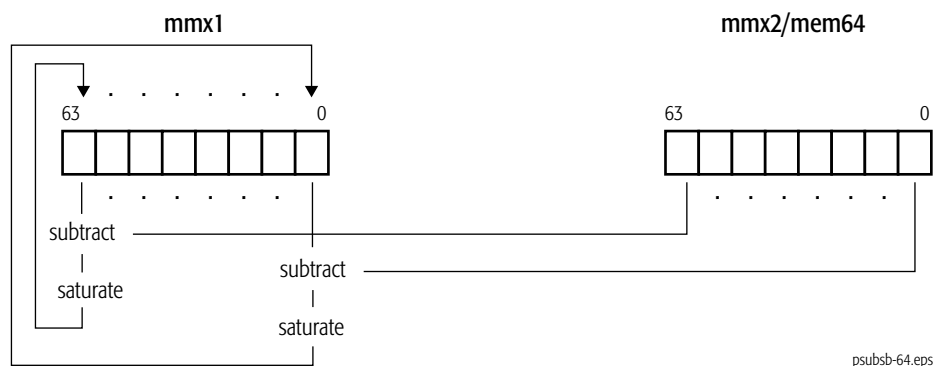
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The SSE-2 bit (bit 26) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBSB**Packed Subtract Signed With Saturation Bytes**

The PSUBSB instruction subtracts each packed 8-bit signed integer value in the second source operand from the corresponding packed 8-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBSB <i>mmx1, mmx2/mem64</i>	OF E8/ <i>r</i>	Subtracts packed byte signed integer values in an MMX register or 64-bit memory location from packed byte integer values in another MMX register and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSW, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

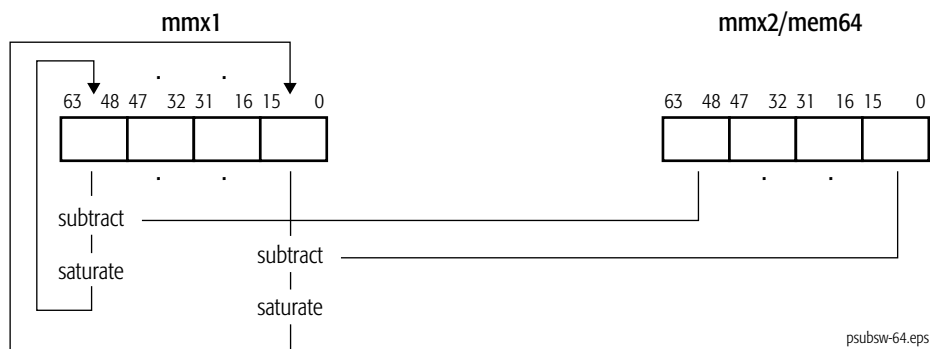
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBSW**Packed Subtract Signed With Saturation Words**

The PSUBSW instruction subtracts each packed 16-bit signed integer value in the second source operand from the corresponding packed 16-bit signed integer in the first source operand and writes the signed integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination and source operands are an MMX register and another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBSW <i>mmx1</i> , <i>mmx2/mem64</i>	0F E9 /r	Subtracts packed 16-bit signed integer values in an MMX register or 64-bit memory location from packed 16-bit integer values in another MMX register and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBUSB, PSUBUSW, PSUBW

rFLAGS Affected

None.

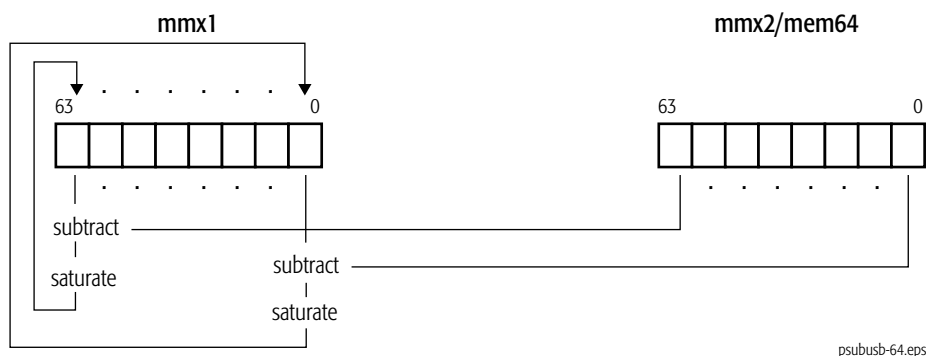
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBUSB**Packed Subtract Unsigned and Saturate Bytes**

The PSUBUSB instruction subtracts each packed 8-bit unsigned integer value in the second source operand from the corresponding packed 8-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding byte of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBUSB <i>mmx1, mmx2/mem64</i>	0F D8 /r	Subtracts packed byte unsigned integer values in an MMX register or 64-bit memory location from packed byte integer values in another MMX register and writes the result in the destination MMX register.



For each packed value in the destination, if the value is larger than the largest unsigned 8-bit integer, it is saturated to FFh, and if the value is smaller than the smallest unsigned 8-bit integer, it is saturated to 00h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSW, PSUBW

rFLAGS Affected

None.

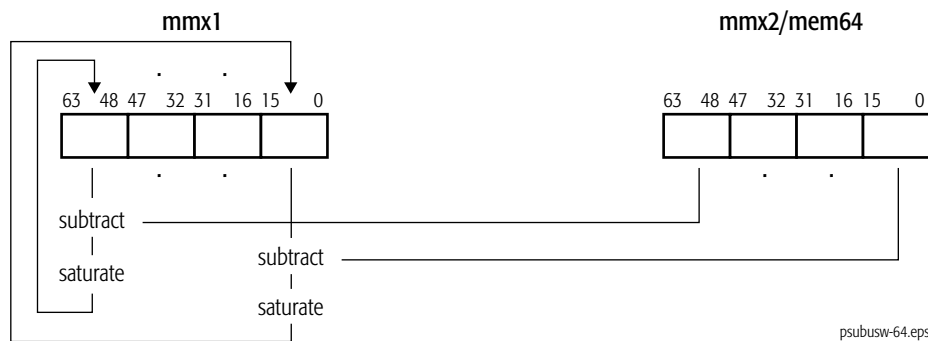
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBUSW**Packed Subtract Unsigned and Saturate Words**

The PSUBUSW instruction subtracts each packed 16-bit unsigned integer value in the second source operand from the corresponding packed 16-bit unsigned integer in the first source operand and writes the unsigned integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBUSW <i>mmx1</i> , <i>mmx2/mem64</i>	0F D9 /r	Subtracts packed 16-bit unsigned integer values in an MMX register or 64-bit memory location from packed 16-bit integer values in another MMX register and writes the result in the destination MMX register.



psubusw-64.eps

For each packed value in the destination, if the value is larger than the largest unsigned 16-bit integer, it is saturated to FFFFh, and if the value is smaller than the smallest unsigned 16-bit integer, it is saturated to 0000h.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBW

rFLAGS Affected

None.

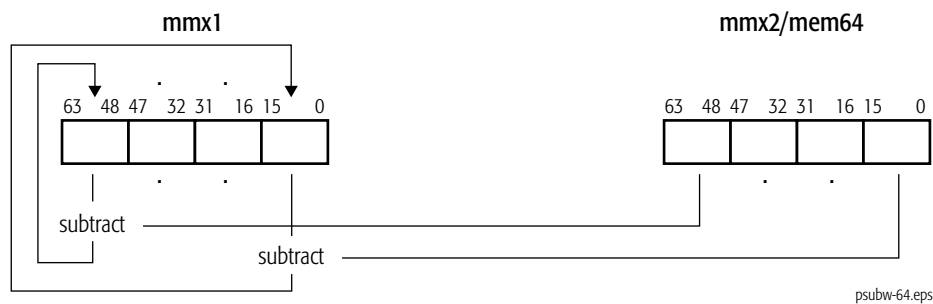
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSUBW**Packed Subtract Words**

The PSUBW instruction subtracts each packed 16-bit integer value in the second source operand from the corresponding packed 16-bit integer in the first source operand and writes the integer result of each subtraction in the corresponding word of the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PSUBW <i>mmx1, mmx2/mem64</i>	0F F9 /r	Subtracts packed 16-bit integer values in an MMX register or 64-bit memory location from packed 16-bit integer values in another MMX register and writes the result in the destination MMX register.



This instruction operates on both signed and unsigned integers. If the result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of the result are written in the destination.

Related Instructions

PSUBB, PSUBD, PSUBQ, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW

rFLAGS Affected

None.

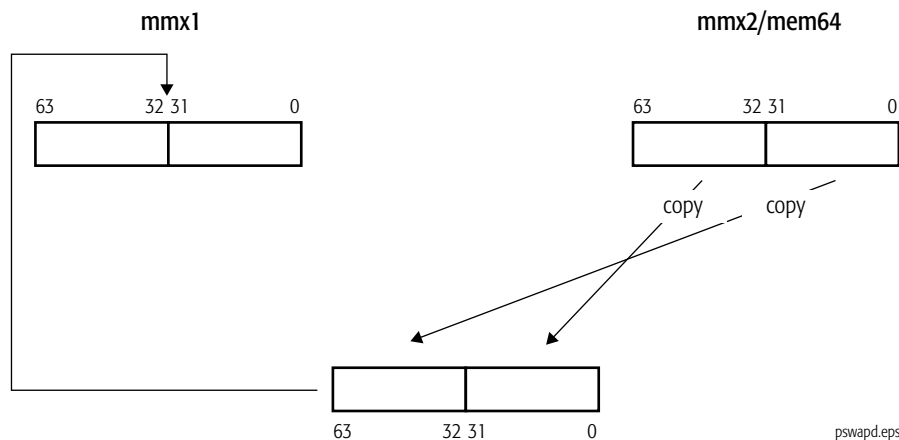
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PSWAPD Packed Swap Doubleword

The PSWAPD instruction swaps (reverses) the two packed 32-bit values in the source operand and writes each swapped value in the corresponding doubleword of the destination. The source operand is an MMX register or 64-bit memory location. The destination is another MMX register.

Mnemonic	Opcode	Description
PSWAPD <i>mmx1, mmx2/mem64</i>	0F 0F BB	Swaps packed 32-bit values in an MMX register or 64-bit memory location and writes each value in the destination MMX register.



Related Instructions

None.

rFLAGS Affected

None.

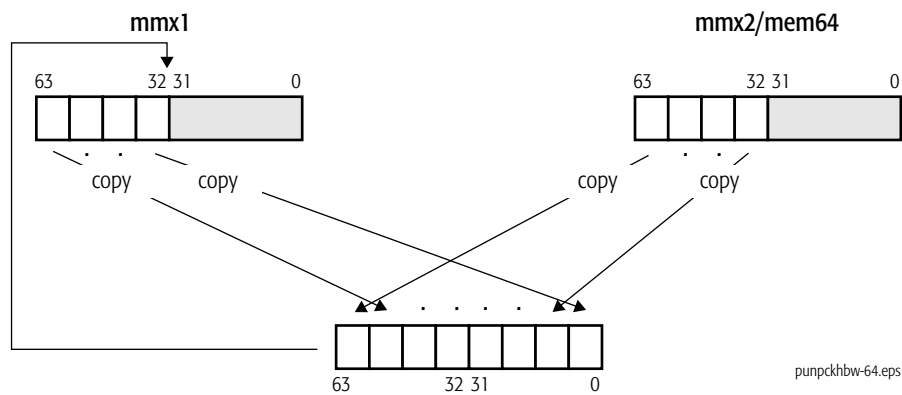
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The AMD Extensions to 3DNow!™ bit (bit 30) in CPUID extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKHBW**Unpack and Interleave High Bytes**

The PUNPCKHBW instruction unpacks the high-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The low-order bytes of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHBW <i>mmx1, mmx2/mem64</i>	0F 68 /r	Unpacks the four high-order bytes in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved bytes in the destination MMX register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

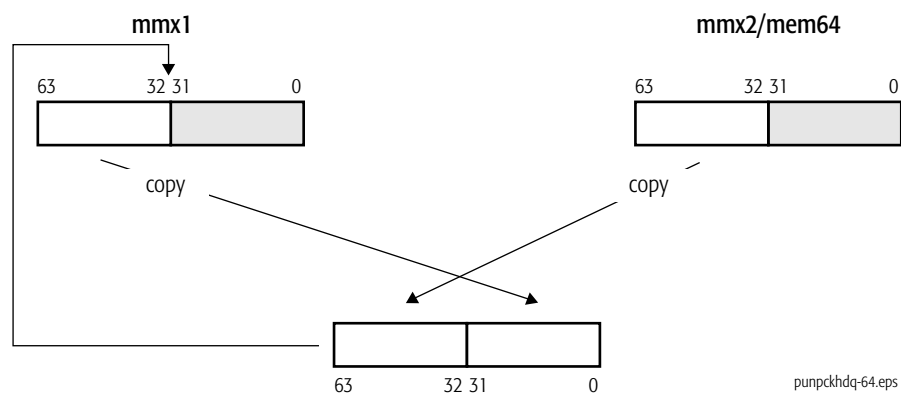
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKHDQ**Unpack and Interleave High Doublewords**

The PUNPCKHDQ instruction unpacks the high-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The low-order doublewords of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHDQ <i>mmx1, mmx2/mem64</i>	0F 6A/r	Unpacks the high-order doubleword in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved doublewords in the destination MMX register.



If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

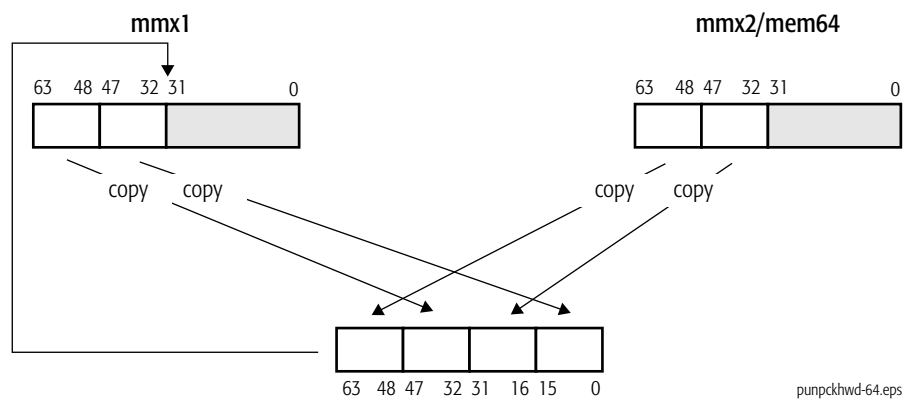
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKHWD**Unpack and Interleave High Words**

The PUNPCKHWD instruction unpacks the high-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The low-order words of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKHWD <i>mmx1, mmx2/mem64</i>	0F 69 /r	Unpacks two high-order words in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved words in the destination MMX register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

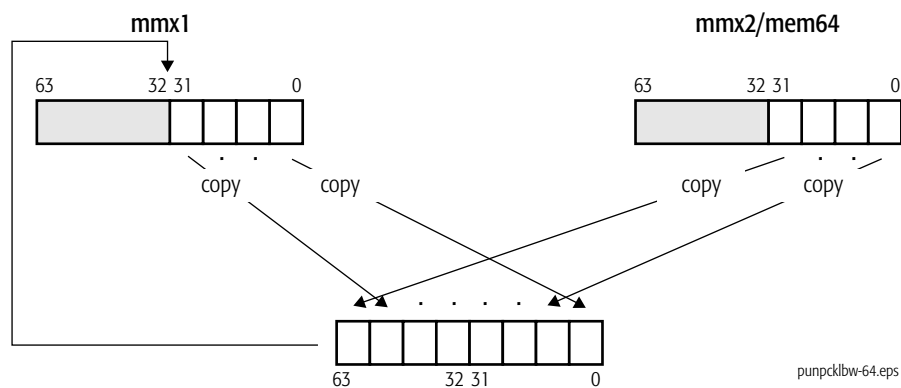
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKLBW**Unpack and Interleave Low Bytes**

The PUNPCKLBW instruction unpacks the low-order bytes from the first and second source operands and packs them into interleaved-byte words in the destination (first source). The high-order bytes of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLBW <i>mmx1, mmx2/mem64</i>	0F 60 /r	Unpacks the four low-order bytes in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved bytes in the destination MMX register.



If the second source operand is all 0s, the destination contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLDQ, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

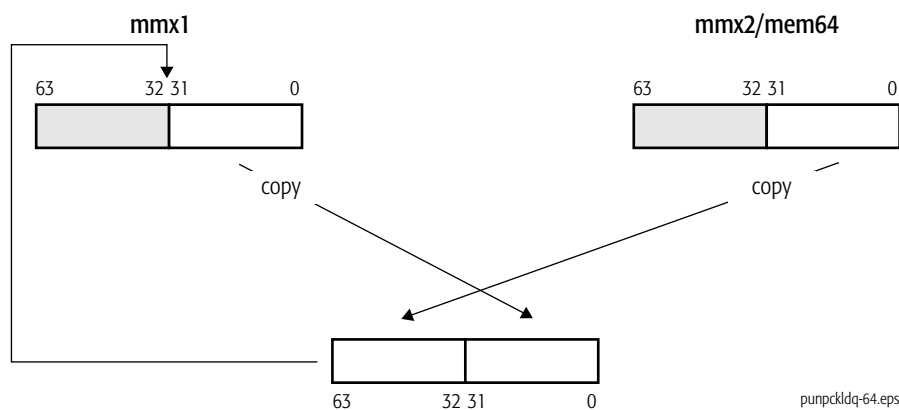
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKLDQ Unpack and Interleave Low Doublewords

The PUNPCKLDQ instruction unpacks the low-order doublewords from the first and second source operands and packs them into interleaved-doubleword quadwords in the destination (first source). The high-order doublewords of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLDQ <i>mmx1, mmx2/mem64</i>	0F 62 /r	Unpacks the low-order doubleword in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved doublewords in the destination MMX register.



If the second source operand is all 0s, the destination contains the doubleword(s) from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLQDQ, PUNPCKLWD

rFLAGS Affected

None.

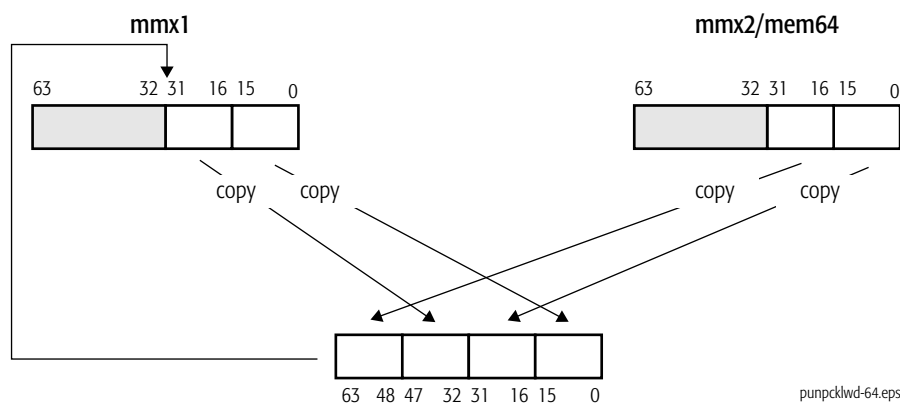
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PUNPCKLWD**Unpack and Interleave Low Words**

The PUNPCKLWD instruction unpacks the low-order words from the first and second source operands and packs them into interleaved-word doublewords in the destination (first source). The high-order words of the source operands are ignored. The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PUNPCKLWD <i>mmx1, mmx2/mem64</i>	0F 61 /r	Unpacks the two low-order words in an MMX register and another MMX register or 64-bit memory location and packs them into interleaved words in the destination MMX register.



If the second source operand is all 0s, the destination contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

Related Instructions

PUNPCKHBW, PUNPCKHDQ, PUNPCKHQDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLQDQ

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

PXOR**Packed Logical Bitwise Exclusive OR**

The PXOR instruction performs a bitwise exclusive OR of the values in the first and second source operands and writes the result in the destination (first source). The first source/destination operand is an MMX register and the second source operand is another MMX register or 64-bit memory location.

Mnemonic	Opcode	Description
PXOR <i>mmx1</i> , <i>mmx2/mem64</i>	0F EF /r	Performs bitwise logical XOR of values in an MMX register and in another MMX register or 64-bit memory location and writes the result in the destination MMX register.

**Related Instructions**

PAND, PANDN, POR

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The emulate bit (EM) of CR0 is set to 1. The MMX bit (bit 23) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

2.9 x87 Floating-Point Instruction Reference

This section describes the function, mnemonic syntax, opcodes, condition codes, affected flags, and possible exceptions generated by the x87 floating-point instructions. The x87 floating-point instructions are used in legacy floating-point applications. Most of these instructions load, store, or operate on data located in the x87 ST(0)–ST(7) stack registers (the FPR0–FPR7 physical registers). The remaining instructions within this category are used to manage the x87 floating-point environment.

The x87 floating-point instructions are supported in a given hardware implementation of the x86-64 architecture if the following CPUID function is set:

- On-Chip Floating-Point Unit, indicated by bit 0 of CPUID standard function 1 and extended function 8000_0001h.
- CMOVcc (conditional moves), indicated by bit 15 of CPUID standard function 1 and extended function 8000_0001h. This bit indicates support for x87 floating-point conditional moves (FCMOVcc) whenever the On-Chip Floating-Point Unit bit (bit 0) is also set.

The x87 instructions can be used in legacy mode or long mode. Their use in long mode is available if the following CPUID function is set:

- Long Mode, indicated by bit 29 of CPUID extended function 8000_0001h.

Compilation of x87 media programs for execution in 64-bit mode offers two primary advantages: access to the 64-bit virtual address space, and access to the RIP-relative addressing mode.

For further information about the x87 floating-point instructions and register resources, see:

- Volume 1: Section 6, “x87 Floating-Point Programming”, on page 289.
- Volume 3: Section 2.3, “Summary of Registers and Data Types”, on page 28.
- Volume 3: Section 2.5, “Notation”, on page 41.
- Volume 3: Section 1.2, “Instruction Prefixes”, on page 3.

F2XM1**Floating-Point Compute 2^x-1**

Calculates the exponential value of 2 to the power of the source operand minus 1. The source operand is located in register ST(0) and the result is also stored in ST(0). The value of the source operand must lie in the range -1.0 to $+1.0$. If the source value is outside this range, the result is undefined.

This instruction, when used in conjunction with the FYL2X instruction, can be applied to calculate $z = xy$ by taking advantage of the log property $x^y = 2^{y \cdot \log_2 x}$.

Mnemonic	Opcode	Description
F2XM1	D9 F0	ST(0) becomes $(2^{\text{ST}(0)} - 1)$.

Related Instructions

FYL2X, FYL2XP1.

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The result is a denormal operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FABS**Floating-Point Absolute Value**

Clears the sign bit of ST(0) to create the absolute value of the operand. The following table shows the results obtained when creating the absolute value of various classes of numbers.

This operation applies even if the value in ST(0) is negative zero or negative infinity.

Mnemonic	Opcode	Description
FABS	D9 E1	Replace ST(0) with its absolute value.

Examples

1. Take the absolute value of a number:

```
FLD Z      ;Put the value Z in ST(0).  
FABS       ;Replace the value in ST(0) with its absolute value.  
FSTP @ABSVAL;Store the result.
```

Related Instructions

FPREM, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FADD

FADDP

FIADD

Floating-Point Add

FADD adds the source and destination operands, and puts the result in the destination. If only one operand is given, the destination is ST(0) and source is ST(*i*) or any 32-bit/64-bit floating-point data variable. If no operands are specified, ST(1) and ST(0) become the source and destination, respectively.

FADDP adds source to destination, places result in the destination register, and pops ST(0). The destination can be any x87 stack register. ST(0) is always the source.

FIADD converts an integer source operand to double-extended-precision format before adding the destination and source operands. It then stores the floating-point sum in the destination location. The destination operand is the ST(0) register. The source operand is a memory location containing a 16-bit or 32-bit integer.

Mnemonic	Opcode	Description
<i>FADD mem32real</i>	D8 /0	ST(0) becomes the sum of ST(0) and 32-bit memory value.
<i>FADD mem64real</i>	DC /0	ST(0) becomes the sum of ST(0) and 64-bit memory value.
<i>FADD ST(0),ST(<i>i</i>)</i>	D8 C0+ <i>i</i>	ST(0) become the sum of ST(0) and ST(<i>i</i>).
<i>FADD ST(<i>i</i>),ST(0)</i>	DC C0+ <i>i</i>	ST(<i>i</i>) become the sum of ST(0) and ST(<i>i</i>).
<i>FADDP</i>	DE C1	ST(1) becomes the sum of ST(0) and ST(1) and the stack is popped.
<i>FADDP ST(<i>i</i>),ST(0)</i>	DE C0+ <i>i</i>	ST(<i>i</i>) become the sum of ST(0) and ST(<i>i</i>) and the stack is popped.
<i>FIADD mem32int</i>	DA /0	ST(0) become the sum of ST(0) and the 32-bit integer in memory.
<i>FIADD mem16int</i>	DE /0	ST(0) becomes the sum of ST(0) and the 16-bit integer in memory.

Related Instructions

FADDP, FIADD

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
	X	X	X	The operands are infinities of unlike sign.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The result is a denormal operand.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FBLD**Floating-Point Load Binary-Coded Decimal**

Converts the BCD source operand into double-extended-precision format and pushes the value onto the x87 stack. The source operand is loaded without rounding errors. The sign of the source operand is preserved, including that of -0 .

The packed BCD digits are assumed to be in the range 0 through 9. The instruction does not check for invalid digits (AH through FH). Attempting to load an invalid encoding produces an undefined result.

Mnemonic	Opcode	Description
FBLD <i>mem80dec</i>	DF /4	Converts a packed BCD value to floating-point and pushes the result onto the x87 stack.

Related Instructions

FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FBSTP Floating-Point Store Binary-Coded Decimal and Pop

Converts the value in the ST(0) register to an 18-digit packed BCD integer, stores the result in the destination operand, and pops the register stack. If the source value is a non-integral value, it is rounded to an integer value, according to rounding mode specified by the RC field of the FPU control word. To pop the register stack, the processor marks the ST(0) register as empty and increments the stack pointer (TOP) by 1.

The destination operand specifies the address where the first byte destination value is to be stored. The BCD value (including its sign bit) requires 10 bytes of space in memory.

Mnemonic	Opcode	Description
FBSTP <i>mem80dec</i>	DF /6	Convert floating-point in ST(0) to BCD, store in mem80 and pop the floating point stack.

Examples

1. Add two packed binary numbers:

```

FBLD 1234567890h ;Push 1234567890 onto ST(0).
FBLD 0987654321h ;Push 0987654321 onto ST(0).
                ;ST(1) is now 1234567890h.
FADD                ;ST(0) is now 84746B83h.
FBSTP SUM          ;SUM now contains the packed BCD number 2222222211h.
                ;ST(0) is 1234567890h.
```

Related Instructions

FBLD

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Source operand is empty or contains a NaN, an unsupported format, or value that exceeds 18 BCD digits in length.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Inexact result (5) (precision)	X	X	X	The result cannot be represented exactly in the destination format.

FCHS**Floating-Point Change Sign**

Complements the sign bit of the value in ST(0). This operation changes a positive value into a negative value of equal magnitude or vice versa. This applies to positive and negative floating point values, as well as -0 and $+0$, NaNs and $+\infty$ and $-\infty$.

Mnemonic	Opcode	Description
FCHS	D9 E0	Reverses the sign bit of ST(0).

Related Instructions

FABS, FPREM, FRNDINT, FXTRACT

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FCLEX
FNCLEX**Floating-Point Clear Flags**

Clears the floating-point exception flags (PE, UE, OE, ZE, DE, and IE), the exception summary status flag (ES), the stack fault flag (SF), and the busy flag (B) in the FPU status word. The four condition-code bits are left undefined.

Assemblers issue FCLEX as an FWAIT instruction (9Bh) followed by an FNCLEX instruction. Thus, FCLEX (but not FNCLEX) reports pending unmasked x87 floating-point exceptions before clearing the exception flags.

Mnemonic	Opcode	Description
FCLEX	9B DB E2	Checks for pending floating-point exceptions and then clears floating point exception flags.
FNCLEX	DB E2	Clear floating-point flags without checking for pending floating-point unmasked exceptions.

Related Instructions

FNCLEX

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FCMOVcc**Floating-Point Conditional Move If Below**

This set of instructions, including the mnemonics FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNE, FCMOVNU and FCMOVU, test the status flags in the rFLAGS register and, depending upon the values encountered, moves a value in the source stack register (second operand) to the destination stack register (first operand). The source operand is may be any stack register. the destination operand is always ST(0).

Mnemonic	Opcode	Description
FCMOVB ST(0),ST(<i>i</i>)	DA C0+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the carry flag is 1.
FCMOVBE ST(0),ST(<i>i</i>)	DA D0+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the carry flag is 1 or the zero flag is 1.
FCMOVE ST(0),ST(<i>i</i>)	DA C8+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the zero flag is 1.
FCMOVNB ST(0),ST(<i>i</i>)	DB C0+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the carry flag is 0.
FCMOVNBE ST(0),ST(<i>i</i>)	DB D0+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the carry flag is 0 and the zero flag is 0.
FCMOVNE ST(0),ST(<i>i</i>)	DB C8+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the zero flag is 0.
FCMOVNU ST(0),ST(<i>i</i>)	DB D8+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the parity flag is 0.
FCMOVU ST(0),ST(<i>i</i>)	DA D8+ <i>i</i>	Move the contents of ST(<i>i</i>) into ST(0) if the parity flag is 1.

Related Instructions

FCMOVB, FCMOVE, FCMOVBE, FCMOVU, FCMOVNB, FCMOVNE, FCMOVBE, FCMOVNU

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The Conditional Move Instructions bit (bit 15) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FCOM Floating-Point Compare

FCOMP

FCOMPP

Compares the value in ST(0) to a source value and sets condition code flags C0, C2, and C3 in the FPU status word depending upon the results.

Mnemonic	Opcode	Description
FCOM	D8 D1	Compare the contents of ST(0) with the contents of ST(1).
FCOM <i>mem32real</i>	D8 /2	Compare the contents of <i>mem32real</i> with the contents ST(0).
FCOM <i>mem64real</i>	DC /2	Compare the contents of <i>mem64real</i> with the contents of ST(0).
FCOM ST(<i>i</i>)	D8 D0+ <i>i</i>	Compare the contents of ST(<i>i</i>) with the contents of ST(0).
FCOMP	D8 D9	Compare the contents of ST(0) with the contents of ST(1) and pop x87 stack.
FCOMP <i>mem32real</i>	D8 /3	Compare the contents of <i>mem32real</i> with the contents ST(0) and pop x87 stack.
FCOMP <i>mem64real</i>	DC /3	Compare the contents of <i>mem64real</i> with the contents of ST(0) and pop x87 stack.
FCOMP ST(<i>i</i>)	D8 D8+ <i>i</i>	Compare the contents of ST(<i>i</i>) with the contents of ST(0) and pop x87 stack.
FCOMPP	DE D9	Compare the contents of ST(0) with the contents of ST(1) and pop x87 stack twice.

The following table shows the resulting settings of the condition code flags.

C3	C2	C0	Description
0	0	0	ST(0) >> source
0	0	1	ST(0) << source
1	0	0	ST(0) = source
1	1	1	numbers cannot be compared

The source operand can be a data register or a memory location. If no source operand is given, the value in ST(0) is compared with the value in ST(1). The sign of zero is ignored, so that $-0.0 = +0.0$.

The FCOM instructions raise an invalid-operation (IE) exception when either or both of the operands is a NaN value or is in an unsupported format. If the exception is masked, the condition flags are set to “unordered.” If the invalid-arithmetic-operand exception is unmasked, the condition code flags are not set.

The FCOMP instruction pops the stack following the comparison operation.

FCOMPP instruction pops the register stack twice following the comparison operation.

The FUCOM/FUCOMP/FUCOMPP instructions perform the same operations as the FCOM/FCOMP/FCOMPP instructions, except that they do not generate an invalid-arithmetic-operand exception for QNaNs.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
<i>U indicates ‘undefined.’</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are NaN values or have unsupported format.
	X	X	X	Register is marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands are denormal values.

FCOMI Floating-Point Compare and Set Flags

FCOMIP

Compares the contents of ST(0) with the contents of another floating-point register. Sets the zero flag (ZF), parity flag (PF) and carry flag (CF), as shown in the table in the x87 Condition Code section.

The FCOMI/FCOMIP instructions set the status flags to “unordered” and generate an invalid-operation exception (IE) when either or both of the operands is a NaN value or is in an unsupported format.

After completion of the instruction, FCOMIP pops the stack.

Mnemonic	Opcode	Description
FCOMI ST(0),ST(<i>i</i>)	DB F0+ <i>i</i>	Compare the contents of ST(0) with the contents of ST(<i>i</i>) and set status flags to reflect the results of the comparison.
FCOMIP ST(0),ST(<i>i</i>)	DF F0+ <i>i</i>	Compare the contents of ST(0) with the contents of ST(<i>i</i>), set status flags to reflect the results of the comparison and pop the x87 stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

Flag	ST(0) > ST(i)	ST(0) < ST(i)	ST(0) = ST(i)	Unordered
ZF	0	0	1	1
PF	0	0	0	1
CF	0	1	0	1

x87 Condition Code

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are NaN values or have unsupported format.
	X	X	X	Register is marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands are denormal values.

FCOS**Floating-Point Cosine**

Calculates the cosine of the source operand in register ST(0) and stores the result in ST(0).

The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. If the source operand is outside this range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range.

The program must check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be changed to an equivalent value within this range by using the FPREM instruction with a divisor of 2π .

Mnemonic	Opcode	Description
FCOS	D9 FF	Replace ST(0) with cosine ST(0).

Related Instructions

FPTAN, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The result is a denormal operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FDECSTP**Floating-Point Decrement Stack-Top Pointer**

Subtracts one from the TOP field of the FPU status word (decrements the top-of-stack pointer). If the TOP field contains a 0, it is set to 7. The effect of this instruction is to rotate the stack by one position. The contents of the x87 data registers and tag register are not affected.

Mnemonic	Opcode	Description
FDECSTP	D9 F6	Decrement the TOP field in the x87 status word.

Data Register	Before FDECSTP			After FDECSTP	
	Value	Stack Pointer		Stack Pointer	Value
7	num1	ST(7)		ST(0)	num1
6	num2	ST(6)		ST(7)	num2
5	num3	ST(5)		ST(6)	num3
4	num4	ST(4)		ST(5)	num4
3	num5	ST(3)		ST(4)	num5
2	num6	ST(2)		ST(3)	num6
1	num7	ST(1)		ST(2)	num7
0	num8	ST(0)		ST(1)	num8

Related Instructions

FINCSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FDIV
FDIVP
FIDIV

Floating-Point Divide

These instructions divide the destination operand by the source operand and store the result in the destination location. The destination operand (dividend) is always in an x87 register. The source operand (divisor) can be a register or a memory location.

The FDIVP instruction pops the stack after performing the division. The no-operand version of the FDIVP instruction divides ST(1) register by the ST(0) register. The FIDIV instruction converts an integer source operand into double-extended-precision floating-point format before performing the division. When the source operand is an integer 0, it is treated as a +0. The one-operand version divides the ST(0) register by the contents of a memory location.

If an unmasked divide by zero exception (#Z) is generated, no result is stored. If the exception is masked, an ∞ of the appropriate sign is stored in the destination operand. The sign of the result determined by the operands, even if an operand is zero.

Mnemonic	Opcode	Description
FDIV <i>mem32real</i>	D8 /6	Divide ST(0) by the contents of <i>mem32real</i> and store result in ST(0).
FDIV <i>mem64real</i>	DC /6	Divide ST(0) by the contents of <i>mem64real</i> and store result in ST(0).
FDIV ST(0),ST(<i>i</i>)	D8 F0+ <i>i</i>	Divide ST(0) by the contents of ST(<i>i</i>) and store result in ST(<i>i</i>).
FDIV ST(<i>i</i>),ST(0)	DC F8+ <i>i</i>	Divide ST(<i>i</i>) by the contents of ST(0) and store result in ST(<i>i</i>).
FDIVP	DE F9	Divide ST(1) by ST(0), store result in ST(1), and pop floating-point register.
FDIVP ST(<i>i</i>),ST(0)	DE F8+ <i>i</i>	Divide ST(<i>i</i>) by ST(0), store result in ST(<i>i</i>) and pop x87 stack.
FIDIV <i>mem32int</i>	DA /6	Divide ST(0) by the contents of <i>mem32int</i> and store result in ST(0).
FIDIV <i>mem64int</i>	DE /6	Divide ST(<i>i</i>) by the contents of <i>mem64int</i> and store result in ST(0).

Related Instructions

FDIVP, FIDIV, FDIVR, FDIVRP, FIDIVR

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper-and exception (DE)	X	X	X	The result is a denormal operand.
Zero-divide exception (ZE)	X	X	X	DEST/±0, where DEST is not equal to ±0.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FDIVR Floating-Point Divide Reverse

FDIVRP

FIDIVR

These instructions divide the source operand by the destination operand and store the result in the destination location. The destination operand (divisor) is always in an x87 register. The source operand (dividend) can be a register or a memory location.

The FDIVR and FDIVRP instruction can take source operands in memory in single-precision or double-precision floating-point format.

The FIDIVR instruction takes source operand in word integer or short integer formats.

The FDIVRP instruction pops the stack after performing the division operation. The no-operand version of the FDIVRP instruction divides the contents of the ST(0) register by the contents of the ST(1) register.

The one-operand version of these instructions divide the contents of a memory location by the contents of the ST(0) register. The two-operand versions divide the contents of the ST(*i*) register by the contents of the ST(0) register or vice versa.

Mnemonic	Opcode	Description
FDIVR <i>mem32real</i>	D8 /7	Divide the contents of <i>mem32real</i> by the contents of ST(0) and store result in ST(0).
FDIVR <i>mem64real</i>	DC /7	Divide the contents of <i>mem64real</i> by the contents of ST(0) and store result in ST(0).
FDIVR ST(0),ST(<i>i</i>)	D8 F8+ <i>i</i>	Divide the contents of ST(<i>i</i>) by the contents of ST(0) and store the result in ST(0).
FDIVR ST(<i>i</i>), ST(0)	DC F0+ <i>i</i>	Divide the contents of ST(0) by the contents of ST(<i>i</i>) and store the result in ST(<i>i</i>).
FDIVRP	DE F1	Divide the contents of ST(0) by the contents of ST(1), store result in ST(1) and pop the floating-point register stack.
FDIVRP ST(<i>i</i>), ST(0)	DE F0 + <i>i</i>	Divide the contents of ST(0) by the contents of ST(<i>i</i>), store the result in ST(<i>i</i>) and pop the floating-point register stack.
FIDIVR <i>mem16int</i>	DE /7	Divide the contents of <i>mem16int</i> by the contents of ST(0) and store result in ST(0).
FIDIVR <i>mem32int</i>	DA /7	Divide the contents of <i>mem32int</i> by the contents of ST(0) and store result in ST(0).

Related Instructions

FDIV, FDIVRP, FIDIV, FIDIVR

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The result is a denormal operand.
Zero-divide exception (ZE)	X	X	X	SRC / ± 0 , where SRC is not equal to ± 0 .
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FFREE**Free Floating-Point Register**

Frees a destination x87 stack register by marking its tag register entry as empty. Neither the contents of the freed register nor stack designations (ST(0)) are affected.

Mnemonic	Opcode	Description
FFREE ST(<i>i</i>)	DD C0+ <i>i</i>	Set tag for x87 stack register <i>i</i> to empty.

Related Instructions

FLD, FST, FSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FICOM FICOMP

Floating-Point Integer Compare

These instructions compare the value in ST(0) with an integer source operand and set the condition code flags C0, C2, and C3 in the FPU status word according to the results (see table below). The source operand can be either a 16-bit or 32-bit integer in memory. The integer value is converted to double-extended-precision format before making the comparison.

The FICOMP instruction pops the stack after performing the comparison.

Mnemonic	Opcode	Description
FICOM <i>mem16int</i>	DE /2	Compare the contents of <i>mem16int</i> to the contents of ST(0).
FICOM <i>mem32int</i>	DA /2	Compare the contents of <i>mem32int</i> to the contents of ST(0).
FICOMP <i>mem16int</i>	DE /3	Compare the contents of <i>mem16int</i> to the contents of ST(0) and pop the floating-point register stack.
FICOMP <i>mem32int</i>	DA /3	Compare the contents of <i>mem32int</i> to the contents of ST(0) and pop the floating-point register stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None.

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) >> source
0	0	1	ST(0) << source
1	0	0	ST(0) = source
1	1	1	numbers cannot be compared

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are NaN values or unsupported formats.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper-and exception (DE)	X	X	X	One or both operands are denormal values.

FILD**Floating-Point Load Integer**

Converts the signed-integer source operand into double-extended-precision format and pushes the value onto the stack. The source operand can be a 16, 32, or 64 bit integer value. It is loaded without rounding errors. The sign of the source operand is preserved.

Mnemonic	Opcode	Description
FILD <i>mem16int</i>	DF /0	Push the contents of <i>mem16int</i> onto the floating-point register stack.
FILD <i>mem32int</i>	DB /0	Push the contents of <i>mem32int</i> onto the floating-point register stack.
FILD <i>mem64int</i>	DF /5	Push the contents of <i>mem64int</i> onto the floating-point register stack.

Related Instructions

FLD, FST, FSTP, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FINCSTP**Floating-Point Increment Stack-Top Pointer**

Adds one to the TOP field of the FPU status word (increments the top-of-stack pointer). If the TOP field contains a 7, it is set to 0. The effect of this instruction is to rotate the stack by one position. The contents of the x87 data registers and tag register are not affected. This operation is not equivalent to popping the stack, because the tag for the previous top-of-stack register is not marked empty.

Mnemonic	Opcode	Description
FINCSTP	D9 F7	Increment the status word stack pointer.

Data Register	Before FINCSTP			After FINCSTP	
	Value	Stack Pointer		Stack Pointer	Value
7	num1	ST(7)		ST(6)	num1
6	num2	ST(6)		ST(5)	num2
5	num3	ST(5)		ST(4)	num3
4	num4	ST(4)		ST(3)	num4
3	num5	ST(3)		ST(2)	num5
2	num6	ST(2)		ST(1)	num6
1	num7	ST(1)		ST(0)	num7
0	num8	ST(0)		ST(7)	num8

Related Instructions

FDECSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FINIT
FNINIT**Floating-Point Initialize**

Sets the FPU control, status, tag, instruction pointer, and data pointer registers to their default states. The FPU control word is set to 037FH (round to nearest, all exceptions masked, 64-bit precision). The status word is cleared (no exception flags set, TOP is set to 0). The data registers in the register stack are left unchanged, but they are all tagged as empty (11B). Both the instruction and data pointers are cleared.

Assemblers issue FINIT as an FWAIT instruction (9Bh) followed by an FNINIT instruction. Thus, FINIT (but not FNINIT) reports pending unmasked x87 floating-point exceptions before performing the initialization.

Mnemonic	Opcode	Description
FINIT	9B DB E3	Initialize the floating-point unit after checking for unmasked floating-point exceptions.
FNINIT	DB E3	Initialize the floating-point unit without checking for unmasked floating-point exceptions.

Related Instructions

FNINIT

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	0	
C1	0	
C2	0	
C3	0	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FIST FISTP

Floating-Point Integer Store

Rounds the value in ST(0) to an integer and copies it to the specified destination in memory. The destination can be a 16-bit, 32-bit, or 64-bit integer in memory. The type of rounding is determined by the rounding control (RC) field of the x87 control word. The default is round-to-nearest.

The FISTP instruction pops the stack after storing.

C0, C1, C2, C3 are cleared to 0.

Mnemonic	Opcode	Description
FIST <i>mem16int</i>	DF /2	Store the contents of ST(0) into <i>mem16int</i> .
FIST <i>mem32int</i>	DB /2	Store the contents of ST(0) into <i>mem32int</i> .
FISTP <i>mem16int</i>	DF /3	Store the contents of ST(0) into <i>mem16int</i> and pop the floating-point register stack.
FISTP <i>mem32int</i>	DB /3	Store the contents of ST(0) into <i>mem32int</i> and pop the floating-point register stack.
FISTP <i>mem64int</i>	DF /7	Store the contents of ST(0) into <i>mem64int</i> and pop the floating-point register stack.

The following table shows the results obtained when storing various classes of numbers in integer format.

ST(0)	DEST
-•	invalid-operation (IE) exception
-FP < -1	-Integer
-1 < -FP < -0	0 or ± 1 , depending on the rounding mode
-0	0
+0	0
+0 < +FP < +1	0 or ± 1 , depending on the rounding mode
+FP > +1	+Integer
+•	invalid-operation (IE) exception
NaN	invalid-operation (IE) exception

Related Instructions

FLD, FST, FSTP, FILD, FISTP, FBLD, FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is too large for the destination format.
	X	X	X	The source operand is a NaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FLD Floating-Point Load

Pushes the source operand onto the stack. If the source operand is in single-precision or double-precision format, it is automatically converted to the double-extended-precision format before being pushed on the stack.

The FLD instruction can also push the value in a selected x87 register, ST(*i*), onto the stack. In this case, pushing register ST(0) duplicates the stack top.

Mnemonic	Opcode	Description
FLD <i>mem32real</i>	D9 /0	Push the contents of <i>mem32real</i> onto the floating-point register stack.
FLD <i>mem64real</i>	DD /0	Push the contents of <i>mem64real</i> onto the floating-point register stack.
FLD <i>mem80real</i>	DB /5	Push the contents of <i>mem80real</i> onto the floating-point register stack.
FLD ST(<i>i</i>)	D9 C0+ <i>i</i>	Push the contents of ST(<i>i</i>) onto the floating-point register stack.

Related Instructions

FFREE, FLD, FST, FSTP, FILD, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

Exception	Real	Virtual 8086	Protected	Cause of Exception
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.
Denormalized-oper-and exception (DE)	X	X	X	The source operand is a denormal value. This exception does not occur if the source operand is in double-extended-precision format.

FLD1 Floating-Point Load +1.0

Pushes the floating-point value +1.0 onto ST(0).

Mnemonic	Opcode	Description
FLD1	D9 E8	Push +1.0 onto the floating-point register stack.

Related Instructions

FLD, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDCW**Floating-Point Load x87 Control Word**

Loads the 16-bit source operand into the FPU control word. The source operand is a memory location. This instruction is typically used to establish or change the x87 mode of operation.

If one or more exception flags are set in the FPU status word prior to loading a new FPU control word and the new control word unmask one or more of those exceptions, a floating-point exception is generated upon execution of the next x87 floating-point instruction (except for the no-wait floating-point instructions.)

To avoid raising exceptions when changing x87 operating modes, clear any pending exceptions before loading the new control word. Use FCLEX or FNCLEX for this purpose.

Mnemonic	Opcode	Description
FLDCW <i>mem2env</i>	D9 /5	Load the contents of <i>mem16</i> into the floating-point control word.

Related Instructions

FSTCW, FNSTCW, FSTSW, FNSTSW, FSTENV, FNSTENV, FLDENV, FCLEX, FNCLEX

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

FLDENV**Floating-Point Load x87 Environment**

Restores an x87 environment stored by the FSTENV instruction. The x87 environment consists of the x87 control word, x87 status word, x87 tag word, last non-control instruction pointer, last data pointer, and opcode of the last non-control instruction completed.

The FLDENV instruction should be executed in the same operating mode as the corresponding FSTENV or FNSTENV instruction.

When executing FLDENV, any exception flags are set in the new status word, and these exceptions are unmasked in the control word, a floating-point exception occurs when the next floating-point instruction (except for the no-wait floating-point instructions) is executed.

To avoid generating exceptions when loading a new environment, clear all the exception flags in the FPU status word that is being loaded. Use FCLEX or FNCLEX for this purpose.

Mnemonic	Opcode	Description
FLDENV <i>mem14/28env</i>	D9 /4	Load the complete contents of the floating-point environment from <i>mem14/28env</i> .

Related Instructions

FSTENV, FNSTENV, FCLEX, FNCLEX

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

FLDL2E Floating-Point Load $\log_2 e$

Pushes $\log_2 e$ onto ST(0). The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the FPU control word.

Mnemonic	Opcode	Description
FLDL2E	D9 EA	Push $\log_2 e$ onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDLG2, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDL2T Floating-Point Load $\log_2 10$

Pushes $\log_2 10$ onto the floating point stack (ST(0)). The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the FPU control word.

Mnemonic	Opcode	Description
FLDL2T	D9 E9	Push $\log_2 10$ onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDLG2 Floating-Point Load $\log_{10} 2$

Pushes $\log_{10} 2$ onto ST(0). The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the FPU control word.

Mnemonic	Opcode	Description
FLDLG2	D9 EC	Push $\log_{10} 2$ onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDLN2 Floating-Point Load Ln 2

Pushes $\log_e 2$ onto ST(0). The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the FPU control word.

Mnemonic	Opcode	Description
FLDLN2	D9 ED	Push $\log_e 2$ onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDPI, FLDL2T, FLDL2E, FLDLG2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDPI**Floating-Point Load Pi**

Pushes π onto ST(0). The value in ST(0) is the result, in double-extended-precision format, of rounding an internal 66-bit constant according to the setting of the RC field in the FPU control word.

Mnemonic	Opcode	Description
FLDPI	D9 EB	Push π onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDZ, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FLDZ**Floating-Point Load +0.0**

Pushes +0.0 onto ST(0).

Mnemonic	Opcode	Description
FLDZ	D9 EE	Push zero onto the floating-point register stack.

Related Instructions

FLD, FLD1, FLDPI, FLDL2T, FLDL2E, FLDLG2, FLDLN2

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow occurred.

FMUL Floating-Point Multiply

FMULP

FIMUL

Multiplies the destination and source operands and stores the product in the destination location. The destination operand is always an x87 data register. The source operand can be an x87 data register or a memory location. Source operands in memory can be in single-precision or double-precision format.

Mnemonic	Opcode	Description
FMUL <i>mem32real</i>	D8 /1	Multiply the contents of <i>mem32real</i> by the contents of ST(0) and store the result in ST(0).
FMUL <i>mem64real</i>	DC /1	Multiply the contents of <i>mem64real</i> by the contents of ST(0) and store the result in ST(0).
FMUL ST(0),ST(<i>i</i>)	D8 C8+ <i>i</i>	Multiply ST(0) by ST(<i>i</i>) and store the result in ST(0).
FMUL ST(<i>i</i>),ST(0)	DC C8+ <i>i</i>	Multiply ST(0) by ST(<i>i</i>) and store the result in ST(<i>i</i>).
FMULP	DE C9	Multiply the contents of ST(1) by the contents of ST(0), store the result in ST(1) and pop the floating-point register stack.
FMULP ST(<i>i</i>),ST(0)	DE C8+ <i>i</i>	Multiply ST(0) by ST(<i>i</i>), store the result in ST(<i>i</i>) and pop the floating-point register stack.
FIMUL <i>mem16int</i>	DE /1	Multiply the contents of <i>mem16int</i> to the contents of ST(0) and store the result in ST(0).
FIMUL <i>mem32int</i>	DA /1	Multiply the contents of <i>mem32int</i> to the contents of ST(0) and store the result in ST(0).

The no-operand version of the instruction multiplies the contents of the ST(1) register by the contents of the ST(0) register and stores the product in the ST(1) register.

The one-operand version multiplies the contents of the ST(0) register by the contents of a memory location and stores the product in the ST(0) register.

The two-operand version, multiplies the contents of the ST(0) register by the contents of the ST(*i*) register, storing the result in the register specified with the destination operand.

The FMULP instructions perform the additional operation of popping the stack after storing the product. The no-operand version of FMUL is equivalent to the FMULP instructions.

The FIMUL instruction converts an integer source operand in a memory location to double-extended-precision format, multiplies it by ST(0), and stores the product in ST(0) register. The source operand is a memory location. Source operands in memory are in short-integer or word-integer formats.

Related Instructions

FMULP, FIMUL

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
	X	X	X	One operand is ± 0 and the other is \pm infinity.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result cannot be represented exactly in the destination format.

FNOP

Floating-Point No Operation

Performs no operation. This instruction only affects the rIP register. It does not otherwise affect the processor context.

Mnemonic	Opcode	Description
FNOP	D9 D0	Perform no operation.

Related Instructions

FWAIT, NOP

rFLAGS Affected

None.

x87 Condition Code

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FNSAVE**Save No-Wait x87 and MMX State**

The FNSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and reinitializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Unlike FSAVE, FNSAVE does not wait for pending unmasked x87 floating-point exceptions to occur. This no-wait instruction should be used when processor interrupts are disabled and x87 instructions might generate an interrupt that would create an endless wait.

For details about the memory image saved by FNSAVE, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FNSAVE <i>mem94/108env</i>	DD /6	Store a copy of the floating-pointing state to <i>mem94/108env</i> without checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples**1. Save environment and stack data:**

```
FNSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FSAVE, FRSTOR, FXSAVE, FXRSTOR

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	0
C1	0
C2	0
C3	0

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

FPATAN Floating-Point Partial Arctangent

Computes $\theta = \arctan(y/x)$. Put x in ST(0) and y in ST(1). θ is written over y in ST(1), the stack is popped, leaving θ in ST(0).

The FPATAN instruction returns the angle between the X axis and the line from the origin to the point (X,Y), where Y (the ordinate) is ST(1) and X (the abscissa) is ST(0). The angle depends on the sign of X and Y independently, not just on the sign of the ratio Y/X. This is because a point (-X,Y) is in the second quadrant, resulting in an angle between $\pi/2$ and π , while a point (X,-Y) is in the fourth quadrant, resulting in an angle between 0 and $-\pi/2$. A point (-X,-Y) is in the third quadrant, giving an angle between $-\pi/2$ and $-\pi$.

There is no restriction on the range of source operands that FPATAN can accept. For example:

$$\text{Arctan } (\infty/1) = \pi/2.$$

$$\text{Arctan } -(\infty/1) = -\pi/2.$$

$$\text{Arctan } -(\infty/\infty) = -3\pi/4.$$

$$\text{Arctan } \infty/\infty = \pi/4.$$

Mnemonic	Opcode	Description
FPATAN	D9 F3	ST(1) becomes $\arctan(\text{ST}(1)/\text{ST}(0))$ the floating-point register stack is popped.

Related Instructions

FCOS, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result cannot be represented exactly in the destination format.

FPREM**Floating-Point Partial Remainder**

The FPREM instruction returns the remainder obtained by dividing ST(0) by ST(1) and stores it in ST(0). The remainder is computed by an iterative subtract-and-shift long division algorithm in which one quotient bit is calculated in each iteration.

If the exponent difference between ST(0) and ST(1) is less than 64, all integer bits of the quotient are calculated, guaranteeing that the remainder returned is less in magnitude than the divisor in ST(1). If the exponent difference is equal to or greater than 64, only a subset of the integer quotient bits, numbering between 32 and 63, are calculated and a partial remainder is returned. The processor indicates the return of a partial remainder by the setting the C2 condition code to 1.

FPREM is useful for reducing the arguments of periodic functions. When reduction is complete, the instruction stores the three least-significant bits of the quotient in the C3, C1, and C0 flags of the FPU status word.

FPREM is supported for software that was written for early x87 coprocessors. Unlike the FPREM1 instruction, FPREM does not calculate the partial remainder as specified in IEEE Standard 754.

Mnemonic	Opcode	Description
FPREM	D9 F8	ST(0) becomes the remainder of the division of ST(0) by ST(1).

```

ExpDiff = Exponent(ST(0)) - Exponent(ST(1));
if (ExpDiff < 0){
    SW.C2 = 0;
    {SW.C0, SW.C3, SW.C1} = 0;
}
else if (ExpDiff < 64){
    Quotient = Floor(ST(0)/ST(1));
    ST(0) = ST(0) - (ST(1) * Quotient);
    SW.C2 = 0;
    {SW.C0, SW.C3, SW.C1} = Quotient mod 8;
}
else{
    N = 32 + (ExpDiff mod 32);
    Quotient = Floor ((ST(0)/ST(1))/2^(ExpDiff-N));
    ST(0) = ST(0) - (ST(1) * Quotient * 2^(ExpDiff-N));
    SW.C2 = 1;
    {SW.C0, SW.C3, SW.C1} = 0;
}

```


Examples

Related Instructions

FPREM1, FABS, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.

FPREM1**Floating-Point Partial Remainder**

FPREM1 returns the remainder obtained by dividing ST(0) by ST(1) and stores it in ST(0). Unlike, FPREM, it rounds the integer quotient to the nearest integer and returns the remainder corresponding to the back multiply of the rounded quotient.

If the exponent difference between ST(0) and ST(1) is less than 64, all integer as well as additional fractional bits of the quotient are calculated to do the rounding. The remainder returned is a complete remainder and is less than or equal to one half of the magnitude of the divisor. If the exponent difference is equal to or greater than 64, only a subset of the integer quotient bits, numbering between 32 and 63, are calculated and a partial remainder is returned. The processor indicates the return of a partial remainder by the setting the C2 condition code to 1.

Rounding control has no effect. FPREM1 results are exact.

Mnemonic	Opcode	Description
FPREM1	D9 F5	ST(0) becomes the IEEE remainder of the division of ST(0) by ST(1).

Action

```

ExpDiff = Exponent(ST(0)) - Exponent(ST(1));
if (ExpDiff < 0){
    SW.C2 = 0;
    {SW.C0, SW.C3, SW.C1} = 0;
}
else if (ExpDiff < 64){
    Quotient = Integer obtained by rounding (ST(0)/ST(1)) to nearest integer
                (if result is exactly halfway between 2 integers, round
                to nearest even integer)
    ST(0) = ST(0) - (ST(1) * Quotient);
    SW.C2 = 0;
    {SW.C0, SW.C3, SW.C1} = Quotient mod 8;
}
else{
    N = 32 + (ExpDiff mod 32);
    Quotient = Floor ((ST(0)/ST(1))/2^(ExpDiff-N));
    ST(0) = ST(0) - (ST(1) * Quotient * 2^(ExpDiff-N));
    SW.C2 = 1;
    {SW.C0, SW.C3, SW.C1} = 0;
}

```

Examples

Related Instructions

FPREM1, FABS, FRNDINT, FXTRACT, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Floating-Point Exception				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.

FPTAN**Floating-Point Partial Tangent**

Computes the tangent of the source operand in register ST(0), stores the result in ST(0), and pushes a 1.0 onto the stack.

The source operand must be given in radians and must be less than $\pm 2^{63}$. Source values outside this range can be reduced to the range of the instruction by using the FPREM instruction with a divisor of 2π .

Mnemonic	Opcode	Description
FPTAN	D9 F2	ST(0) becomes tangent ST(0) and 1.0 is pushed onto the floating-point register stack.

Related Instructions

FCOS, FPATAN, FSIN, FSINCOS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)				A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FRNDINT**Floating-Point Round to Integer**

Rounds the source value in the ST(0) register to the nearest integral value, depending on the current rounding mode (setting of the RC field of the FPU control word), and stores the result in ST(0).

If the source value is infinity, the value is not changed. If the source value is not an integral value, the floating-point inexact-result exception (#P) is generated.

Mnemonic	Opcode	Description
FRNDINT	D9 FC	Round the contents of ST(0) to integer.

Related Instructions

FABS, FPREM, FXTRACT, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Precision exception (PE)	X	X	X	The source operand is not an integral value.

FRSTOR**Restore x87 and MMX State**

The FRSTOR instruction restores complete x87 state from a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit. The MMX state is also restored by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are loaded from memory.

Unlike the FXRSTOR instruction, FRSTOR checks for pending unmasked x87 floating-point exceptions in the restored image. However, if such a pending unmasked exception is found, the exception does not occur immediately after execution of an FRSTOR instruction. Instead, the processor asserts the FERR# (floating-point error) output signal. The normal rules for x87 exceptions then apply, just as if the exception was created by an x87 instruction. If CRO.NE = 1, the processor vectors to the x87 floating-point exception (#MF) handler at the next non-wait x87 instruction. Otherwise, the processor examines the state of the IGNNE# (ignore numeric error) input signal. If IGNNE# is asserted, the processor executes the next x87 instruction. If IGNNE# is negated, the processor freezes at that instruction boundary until an external interrupt occurs that redirects control to a handler that clears the exception.

For details about the memory image restored by FRSTOR, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FRSTOR <i>mem94/108env</i>	DD /4	Load a copy of the floating-pointing state from <i>mem94/108env</i> .

Examples**1. Save environment and stack data:**

```

FSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr      ;Restore the old operating environment.
```

Related Instructions

FSAVE, FNSAVE, FXSAVE, FXRSTOR

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Loaded from memory.
C1	Loaded from memory.
C2	Loaded from memory.
C3	Loaded from memory.

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)
x87 floating-point exception pending, #MF	X	X	X	An exception is pending due to an x87 floating-point instruction.

FSAVE**Save x87 and MMX State**

The FSAVE instruction writes the complete x87 state to a 94-byte or 108-byte area in memory, depending upon whether the processor is operating in real or protected mode and whether the operand-size attribute is 16-bit or 32-bit, and initializes the x87 state. The MMX state is also saved by virtue of the fact that MMX state shares part of the x87 state. The x87 condition codes are cleared after being saved.

Assemblers issue FSAVE as an FWAIT instruction (9Bh) followed by an FNSAVE instruction. Thus, FSAVE (but not FNSAVE) reports pending unmasked x87 floating-point exceptions before saving the state.

For details about the memory image saved by FSAVE, see Section 11.4, “Media and x87 Processor State”, on page 352 of Volume 2.

Mnemonic	Opcode	Description
FSAVE <i>mem94/108env</i>	9B DD /6	Store a copy of the floating-pointing state to <i>mem94/108env</i> after checking for pending floating-point exceptions, then reinitialize the x87 state.

Examples**1. Save environment and stack data:**

```
FSAVE memory94Ptr      ;Save the x87 environment.
.
.      ;Do something in the clean operating environment.
.
FRSTOR memory94Ptr     ;Restore the old operating environment.
```

Related Instructions

FNSAVE, FRSTOR, FXSAVE, FXRSTOR

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	0	
C1	0	
C2	0	
C3	0	
<i>U indicates 'undefined.'</i>		

Exceptions (All Modes)

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

FSCALE Floating-Point Scale

Replaces the floating-point value ST(0) with $ST(0) * 2^n$, where n is the nearest integer to the floating-point value in ST(1). This provides a fast method of multiplying by integral powers of 2.

In most cases, only the exponent is changed and the mantissa (significand) remains unchanged. However, when the value being scaled in ST(0) is a denormal value, the mantissa is also changed and the result may turn out to be a normalized number. Similarly, if overflow or underflow results from a scale operation, the resulting mantissa will differ from the mantissa source.

The FSCALE instruction can also be used to reverse the action of the FXTRACT instruction.

Mnemonic	Opcode	Description
FSCALE	D9 FD	SCALE ST(0) by ST(1).

Related Instructions

FSQRT, FPREM, FPREM1, FRNDINT, FXTRACT, FABS, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FSIN**Floating-Point Sine**

Calculates the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$ radians.

If the source operand is outside this range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. The program must check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be changed to an equivalent value within this range by using the FPREM instruction with a divisor of 2π .

Mnemonic	Opcode	Description
FSIN	D9 FE	Replace ST(0) with the sine of ST(0).

Related Instructions

FCOS, FPATAN, FSINCOS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FSINCOS

Floating-Point Sine and Cosine

Computes both sine and cosine of ST(0). Places the sine in ST(0) and the cosine in ST(1). The source operand must be within the range -2^{63} to $+2^{63}$ radians.

If the source operand is outside this range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged. The instruction does not raise an exception when the source operand is out of range. The program must check the C2 flag for out-of-range conditions. Source values outside the range -2^{63} to $+2^{63}$ can be changed to an equivalent value within this range by using the FPREM instruction with a divisor of 2π .

Mnemonic	Opcode	Description
FSINCOS	D9 FB	Replace ST(0) with the sine of ST(0) and push cosine of ST(0) onto the floating-point register stack.

Related Instructions

FCOS, FPATAN, FPTAN, FSIN

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result cannot be represented exactly in the destination format.

FSQRT**Floating-Point Square Root**

Calculates the square root of the source value in the ST(0) register and stores the result in ST(0).

Mnemonic	Opcode	Description
FSQRT	D9 FA	Replace ST(0) with the square root of ST(0).

Related Instructions

FSCALE, FPREM, FPREM1, FRNDINT, FXTRACT, FABS, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
	X	X	X	Source operand is a negative value (except for -0).
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FST**Floating-Point Store Stack Top**

The FST instruction copies the value in the ST(0) register to the destination operand, which can be a memory location or another register in the stack. When storing the value in memory, the value is converted to single-precision or double-precision format. If the top-of-stack is a NaN or infinity, FST truncates the stack-top exponent and significand to fit the destination size.

Mnemonic	Opcode	Description
FST <i>mem32real</i>	D9 /2	Copy the contents of ST(0) to <i>mem32real</i> .
FST <i>mem64real</i>	DD /2	Copy the contents of ST(0) to <i>mem64real</i> .
FST ST(<i>i</i>)	DD D0+ <i>i</i>	Copy the contents of ST(0) to ST(<i>i</i>).

Related Instructions

FFREE, FLD, FST, FSTP, FILD, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FSTCW Floating-Point Store Control Word

FNSTCW

Stores the current value of the FPU control word at the specified destination in memory. The FSTCW instruction checks for and handles pending unmasked floating-point exceptions before storing the control word. The FNSTCW instruction does not.

Assemblers issue FSTCW as an FWAIT instruction (9Bh) followed by an FNSTCW instruction. Thus, FSTCW (but not FNSTCW) reports pending unmasked x87 floating-point exceptions before storing the control word.

Mnemonic	Opcode	Description
FSTCW <i>mem2env</i>	9B D9 /7	Copy the floating-point control word to <i>mem2env</i> after checking for pending floating-point exceptions
FNSTCW <i>mem2env</i>	D9 /7	Copy the floating-point control word to the AX register after checking for pending floating-point exceptions.

Related Instructions

FSTSW, FNSTSW, FNSTCW, FSTENV, FNSTENV

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

FSTENV Floating-Point Store x87 Environment

FNSTENV

Stores the current x87 environment to a 14-byte or 28-byte destination address, depending on the whether the processor is operating in protected or real mode and whether the operand-size attribute is 16-bit or 32-bit. The x87 environment consists of the x87 control word, x87 status word, x87 tag word, last non-control instruction pointer, last data pointer, and opcode of the last non-control instruction completed.

Assemblers issue FSTENV as an FWAIT instruction ((9Bh) followed by an FNSTENV instruction. Thus, FSTENV (but not FNSTENV) reports pending unmasked x87 floating-point exceptions before storing the status word. The stored image reflects the x87 state after all floating-point instructions preceding the FNSTENV instruction in the instruction stream have been executed.

This no-wait FNSTENV instruction should be used when processor interrupts are disabled and x87 instructions might generate an interrupt that would create an endless wait.

These instructions are often used by exception handlers because they provide access to the x87 instruction and data pointers. The environment is typically saved in the stack. Masking all exceptions after saving the environment prevents floating-point exceptions from interrupting the exception handler.

This instruction is prefixed with a wait state and should be used when processor interrupts are enabled.

Mnemonic	Opcode	Description
FSTENV <i>mem14/28env</i>	9B D9 /6	Copy the floating-point environment to <i>mem14/28env</i> after checking for pending floating-point exceptions which are then masked.
FNSTENV <i>mem14/28env</i>	D9 /6	Copy the floating-point environment to <i>mem14/28env</i> without checking for pending floating-point exceptions, which are subsequently masked.

Related Instructions

FSTSW, FNSTSW, FSTCW, FNSTCW, FNSTENV

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

FSTP**Floating-Point Store Stack Top and Pop**

Copies ST(0) to a specified destination and pops ST(0). The destination can be ST(*i*), or a single-precision, double-precision, or double-extended-precision value in memory.

FSTP ST(0) is equivalent to popping the stack with no data transfer.

If the destination size is single-precision or double-precision, the significand of the value being stored is rounded to the width of the destination (according to rounding mode specified by the RC field of the FPU control word), and the exponent is converted to the width and bias of the destination format. If the value being stored is too large for the destination format, an overflow exception (OE) is generated and, if the exception is unmasked, no value is stored in the destination operand. If the value being stored is a denormal value, the denormalized-operand exception (DE) is not generated. This condition is simply signaled as an underflow exception (UE).

If the value being stored is ± 0 , $\pm\text{infinity}$, or a NaN, the least-significant bits of the significand and the exponent are truncated to fit the destination format.

Mnemonic	Opcode	Description
FSTP <i>mem32real</i>	D9 /3	Copy the contents of ST(0) to <i>mem32real</i> and pop the floating-point register stack.
FSTP <i>mem64real</i>	DD /3	Copy the contents of ST(0) to <i>mem64real</i> and pop the floating-point register stack.
FSTP <i>mem80real</i>	DB /7	Copy the contents of ST(0) to <i>mem80real</i> and pop the floating-point register stack.
FSTP ST(<i>i</i>)	DD D8+ <i>i</i>	Copy the contents of ST(0) to ST(<i>i</i>) and pop the floating point register stack.

Related Instructions

FFREE, FLD, FST, FSTP, FILD, FIST, FISTP, FBLD, FBSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1. Otherwise, no other exception flags are set and operand is double-extended-precision format.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

Floating-Point Store x87 Status Word

Stores the current value of the FPU control word at the specified destination in memory. The destination operand can be either a two-byte memory location or the AX register.

Assemblers issue FSTSW as an FWAIT instruction (9Bh) followed by an FNSTSW instruction. Thus, FSTSW (but not FNSTSW) reports pending unmasked x87 floating-point exceptions before storing the status word.

The FNSTSW AX form of the instruction is used primarily in conditional branching where the direction of the branch depends on the x87 condition codes. This instruction can also be used to invoke exception handlers (by examining the exception flags) in environments that do not use interrupts. When the FNSTSW AX instruction is executed, the AX register is updated before the processor executes any further instructions.

This no-wait instruction should be used when processor interrupts are disabled and x87 instructions might generate an interrupt that would create an endless wait.

Mnemonic	Opcode	Description
FSTSW <i>mem2env</i>	9B DD /7	Copy the floating-point status word to <i>mem12byte</i> after checking for pending floating-point exceptions.
FSTSW AX	9B DF E0	Copy the floating-point status word to the AX register after checking for pending floating-point exceptions.
FNSTSW <i>mem2env</i>	DD /7	Copy the floating-point status word to <i>mem12byte</i> without checking for pending floating-point exceptions.
FNSTSW AX	DF E0	Copy the floating-point status word to the AX register without checking for pending floating-point exceptions.

Related Instructions

FNSTSW, FSTCW, FNSTCW, FSTENV, FNSTENV

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)

FSUB Floating-Point Subtract

Subtracts *source* from *destination*, and stores the result in *destination*.

The no-operand version of the instruction subtracts the contents of the ST(0) register from the ST(1) register and stores the result in ST(1). The one-operand version subtracts the contents of a memory location (either a floating-point or integer value) from the contents of the ST(0) register and stores the result in ST(0). The two-operand version, subtracts the contents of the ST(0) register from the ST(i) register or vice versa.

The FSUBP instructions perform the additional operation of popping the x87 register stack following the subtraction. To pop the register stack, the processor marks the ST(0) register as empty and increments the stack pointer (TOP) by 1. The no-operand version of the floating-point subtract instructions always results in the register stack being popped. In some assemblers, the mnemonic for this instruction is FSUB rather than FSUBP.

The FISUB instruction converts an integer source operand to double-extended-precision format before performing the subtraction.

Mnemonic	Opcode	Description
FSUB <i>mem32real</i>	D8 /4	ST(0) becomes ST(0) - <i>mem32real</i> .
FSUB <i>mem64real</i>	DC /4	ST(0) becomes ST(0) - <i>mem64real</i> .
FSUB ST(0),ST(<i>i</i>)	D8 E0+ <i>i</i>	ST(0) becomes ST(0) - ST(<i>i</i>).
FSUB ST(<i>i</i>),ST(0)	DC E8+ <i>i</i>	ST(<i>i</i>) becomes ST(<i>i</i>) - ST(0)
FSUBP	DE E9	ST(1) becomes ST(1) - ST(0) and pop x87 stack.
FSUBP ST(<i>i</i>),ST(0)	DE E8+ <i>i</i>	ST(<i>i</i>) becomes ST(<i>i</i>) - ST(0), and pop x87 stack.
FISUB <i>mem16int</i>	DE /4	ST(0) becomes ST(0) - <i>mem16int</i> .
FISUB <i>mem32int</i>	DA /4	ST(0) becomes ST(0) - <i>mem32int</i> .

Related Instructions

FSUBRP, FISUBR, FSUBP, FSUBR, FISUB

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
	X	X	X	The operands are infinities of like sign.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper-and exception (DE)	X	X	X	The source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Inexact result (5) (precision)	X	X	X	The result cannot be represented exactly in the destination format.

FSUBR Floating-Point Subtract Reverse

FSUBRP

FISUBR

Subtracts the destination operand from the source operand and stores the difference in the destination location. The destination operand is always an x87 register. The source operand can be a register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format.

The one-operand version subtracts the contents of the ST(0) register from the contents of a memory location (either a floating-point or integer value) and stores the result in ST(0). The two-operand version, subtracts the contents of ST(*i*) from ST(0) or vice versa.

The FSUBRP instruction subtracts the contents of ST(*i*) from the contents of ST(0), stores the result in ST(*i*) and pops the x87 stack. If there are no operands ST(0) is subtracted from ST(1) with the result in ST(1). The stack is then popped.

The no-operand version of the floating-point reverse subtract instructions always results in the register stack being popped. In some assemblers, the mnemonic for this instruction is FSUBR rather than FSUBRP.

The FISUBR instruction converts an integer source operand to double-extended-precision format before performing the subtraction.

These instructions perform the reverse operations of the FSUB instruction.

Mnemonic	Opcode	Description
FSUBR <i>mem32real</i>	D8 /5	ST(0) becomes <i>mem32real</i> - ST(0).
FSUBR <i>mem64real</i>	DC /5	ST(0) becomes <i>mem64real</i> - ST(0).
FSUBR ST(0),ST(<i>i</i>)	D8 E8+ <i>i</i>	ST(0) becomes ST(<i>i</i>) - ST(0).
FSUBR ST(<i>i</i>),ST(0)	DC E0+ <i>i</i>	ST(<i>i</i>) becomes ST(0) - ST(<i>i</i>)
FSUBRP	DE E1	ST(1) becomes ST(0) - ST(1) and pop x87 stack.
FISUBR <i>mem16int</i>	DE /5	ST(0) becomes <i>mem16int</i> - ST(0).
FISUBR <i>mem32int</i>	DA /5	ST(0) becomes <i>mem32int</i> - ST(0).
FSUBRP ST(<i>i</i>),ST(0)	DE E0+ <i>i</i>	ST(<i>i</i>) becomes ST(0) - ST(<i>i</i>) and pop x87 stack.

Related Instructions

FSUBRP, FISUBR, FSUB, FSUBP, FISUB

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
	X	X	X	The operands are infinities of like sign.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	The source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FTST**Floating-Point Test with Zero**

Compares ST(0) with zero, and sets the condition code according to the x86 Condition Code table below.

Mnemonic	Opcode	Description
FTST	D9 E4	Compare ST(0) to 0.0.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None.

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > source
0	0	1	ST(0) < source
1	0	0	ST(0) = source
1	1	1	numbers cannot be compared

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are NaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	One or both operands are denormal value.

FUCOM Floating-Point Unordered Compare

FUCOMP

FUCOMPP

FUCOM family of instruction compares ST(0) to the source register. If no source is specified, ST(0) is compared to ST(1). x87 condition codes are set, depending upon the outcome the comparison.

After FUCOMP, ST(0) is popped. After FUCOMPP, ST(0) is popped twice.

The FUCOM instructions perform the same operations as the FCOM instructions, but raise the invalid-operation (IE) exception only when either or both operands are an SNaN or are in an unsupported format. QNaNs cause the condition code flags to be set to unordered, but do not cause an exception to be generated. The FCOM instructions raise an invalid-operation exception when either or both of the operands are a NaN value of any kind or are in an unsupported format.

Mnemonic	Opcode	Description
FUCOM	DD E1	Compare ST(0) to ST(1).
FUCOM ST(<i>i</i>)	DD E0+ <i>i</i>	Compare ST(0) to ST(<i>i</i>).
FUCOMP	DD E9	Compare ST(0) to ST(1) and pop the floating-point register stack.
FUCOMP ST(<i>i</i>)	DD E8+ <i>i</i>	Compare ST(0) to ST(<i>i</i>) and pop the floating-point register stack.
FUCOMPP	DA E9	Compare ST(0) to ST(1) and pop the floating-point register stack twice.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None.

x87 Condition Code

C3	C2	C0	Description
0	0	0	ST(0) > ST(i)
0	0	1	ST(0) < ST(i)
1	0	0	ST(0) = ST(i)
1	1	1	numbers cannot be compared

x87 Condition Code	Description
C0	Undefined
C1	0 if numeric underflow is set.
	0 indicates no round up, if inexact result bit is 1.
	1 indicates round up, if inexact result bit is 1.
C2	Undefined
C3	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are SNaN values or have unsupported formats. Detection of a QNaN value in and of itself does not raise an invalid operand exception.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	One or both operands are denormal values.

FUCOMI FUCOMIP

Floating-Point Unordered Compare and Set Flags

Compares the contents of ST(0) with the contents of another floating-point register. Sets the zero flag (ZF), parity flag (PF) and carry flag (CF), as shown in the table in the x87 Condition Code section.

The FUCOMI and FUCOMIP instructions do not generate an invalid-arithmetic-operand exception for QNaNs.

After completion of the instruction, FUCOMIP pops the x87 register stack.

Mnemonic	Opcode	Description
FUCOMI ST(0),ST(<i>i</i>)	DB E8+ <i>i</i>	Compare the contents of ST(0) with the contents of ST(<i>i</i>) and set status flags to reflect the results of the comparison.
FUCOMIP ST(0),ST(<i>i</i>)	DF E8+ <i>i</i>	Compare the contents of ST(0) with the contents of ST(<i>i</i>), set status flags to reflect the results of the comparison and pop the x87 stack.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

Flag	ST(0) > ST(i)	ST(0) < ST(i)	ST(0) = ST(i)	Unordered
ZF	0	0	1	1
PF	0	0	0	1
CF	0	1	0	1

x87 Condition Code

x87 Condition Code	Value	Description
C0	0/1	See previous chart for interpretation.
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	0/1	See previous chart for interpretation.
C3	0/1	See previous chart for interpretation.
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
Stack, #SS	X	X	X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS).
			X	There is a null segment selector in a data-segment register that is accessing memory.
General protection, segment overrun, #GP	X	X		One of the instruction data operands falls outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and both the alignment mask bit (AM) of control register (CR0) and the alignment check bit (AC) of rFLAGS are set to 1. (In Protected Mode, CPL = 3.)
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	One or both operands are NaN values or have unsupported format.
	X	X	X	Register is marked empty.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-operand exception (DE)	X	X	X	One or both operands are denormal values.

FWAIT**Wait for Unmasked x87 Floating-Point Exceptions**

Causes the processor to check for and handle pending, unmasked, floating-point exceptions before proceeding. (FWAIT is an alternate mnemonic for the WAIT). Exception-condition reporting (i.e., assertion of the FERR# output signal) occurs immediately when an error occurs, although the servicing of such an exception occurs when the next x87 non-control instruction is executed (if CR0.NE=1) or when a later instruction, depending on chip-set delay, is executed (if CR0.NE=0).

This instruction is useful for synchronizing exceptions in critical sections of code. Coding a WAIT instruction after a floating-point instruction insures that any unmasked floating-point exceptions are handled before the processor can modify the instruction results.

Mnemonic	Opcode	Description
FWAIT	9B	Check for any outstanding x87 floating-point exceptions.

Related Instructions

WAIT

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	U	
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	MP and TS in CR0 are set.

FXAM **Floating-Point Examine**

Examines the contents of the ST(0) register and sets the condition code flags C0, C2, and C3 in the FPU status word to indicate the class of value or number in the register.

Mnemonic	Opcode	Description
FXAM	D9 E5	Characterize the number in the ST(0) register.

Related Instructions

FCOM, FCOMPP, FCOMI, FCOMIP, FICOM, FICOMP, FTST, FUCOMI, FUCOMIP, FXAM

rFLAGS Affected

None.

x87 Condition Code

C3	C2	C1	C0	Meaning
0	0	0	0	+unnormal
0	0	0	1	+NAN (not a number)
0	0	1	0	-unnormal
0	0	1	1	-NAN
0	1	0	0	+normal
0	1	0	1	+infinite
0	1	1	0	-normal
0	1	1	1	-infinite
1	0	0	0	+0
1	0	0	1	empty
1	0	1	0	-0
1	0	1	1	empty
1	1	0	0	+denormal
1	1	0	1	empty
1	1	1	0	-denormal
1	1	1	1	empty

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.

FXCH Floating-Point Exchange

Exchanges the contents of registers ST(0) and ST(*i*). If no source operand is specified, the contents of ST(0) and ST(1) are exchanged.

This instruction provides a simple means of moving values in the x87 register stack to the top of the stack, ST(0)], so that they can be operated on by those floating-point instructions that can only operate on values in ST(0).

For example, the following instruction sequence

Mnemonic	Opcode	Description
FXCH	D9 C9	Exchange the contents of ST(0) and ST(1).
FXCH ST(<i>i</i>)	D9 C8+ <i>i</i>	Exchange the contents of ST(0) and ST(<i>i</i>).

Related Instructions

FLD, FST, FSTP

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.

FXRSTOR**Restore XMM, MMX, and x87 State**

The FXRSTOR instruction restores the XMM, MMX, and x87 state. The data loaded from memory is the state information previously saved using the FXSAVE instruction. Restoring data with FXRSTOR that had been previously saved with an FSAVE (rather than FXSAVE) instruction results in an incorrect restoration.

Mnemonic	Opcode	Description
FXRSTOR <i>mem512env</i>	OF AE /1	Restores XMM, MMX, and x87 state.

Unlike the FRSTOR instruction, FXRSTOR does not cause pending unmasked x87 floating-point exceptions in the restored image to be recognized when internal control of x87 exceptions is enabled (CR0.NE=1). When loading a new environment, use an FWAIT instruction after the FXRSTOR instruction to check for and handle any pending unmasked x87 exceptions.

If the restored MXCSR register contains a set bit in an exception status flag, and the corresponding exception mask bit is cleared (indicating an unmasked exception), loading the MXCSR register from memory does not cause a SIMD floating-point exception (#XM).

FXRSTOR executes faster than FRSTOR because it does not restore the x87 error pointers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXRSTOR, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXRSTOR instruction. If software running in 64-bit mode executes an FXRSTOR with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXRSTOR with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, the saved image of XMM0–XMM7 and MXCSR is not loaded into the processor. A general-protection exception occurs if there is an attempt to load a non-zero value to the bits in MXCSR that are defined as reserved and cleared (bits 31-16 and bit 6).

Related Instructions

FWAIT, FXSAVE

rFLAGS Affected

None.

x87 Condition Code

None affected.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

FXSAVE Save XMM, MMX, and x87 State

The FXSAVE instruction saves the XMM, MMX, and x87 state. A memory location that is not aligned on a 16-byte boundary causes a general-protection exception, or, in some cases, an alignment-check exception.

Mnemonic	Opcode	Description
FXSAVE <i>mem512env</i>	0F AE /0	Saves XMM, MMX, and x87 state in 512-byte memory location.

Unlike FSAVE and FNSAVE, FXSAVE does not alter the x87 tag bits. The contents of the saved MMX/x87 data registers are retained, thus indicating that the registers may be valid (or whatever other value the x87 tag bits indicated prior to the save). To invalidate the contents of the MMX/x87 data registers after FXSAVE, software must execute an FINIT instruction. Also, FXSAVE (like FNSAVE) does not check for pending unmasked x87 floating-point exceptions. An FWAIT instruction can be used for this purpose.

FXSAVE executes faster than FSAVE or FNSAVE because it does not save the x87 pointer registers (last instruction pointer, last data pointer, and last opcode) except in the relatively rare cases in which the exception-summary (ES) bit in the x87 status word is set to 1, indicating that an unmasked x87 exception has occurred.

The architecture supports two memory formats for FXSAVE, a 512-byte 32-bit legacy format and a 512-byte 64-bit format. Selection of the 32-bit or 64-bit format is accomplished by using the corresponding effective operand size in the FXSAVE instruction. If software running in 64-bit mode executes an FXSAVE with a 32-bit operand size (no REX-prefix operand-size override), the 32-bit legacy format is used. If software running in 64-bit mode executes an FXSAVE with a 64-bit operand size (requires REX-prefix operand-size override), the 64-bit format is used. For details about the memory image restored by FXRSTOR, see Section 11.4.1, “Saving Media and x87 Processor State”, on page 356 of Volume 2.

If the operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0, FXSAVE does not save the image of XMM0–XMM7 and MXCSR. For details about the CR4.OSFXSR bit, see “FXSAVE/FXRSTOR Support (OSFXSR) Bit” on page 63 of Volume 2.

Related Instructions

FINIT, FNSAVE, FRSTOR, FSAVE, FXRSTOR, LDMXCSR, STMXCSR

rFLAGS Affected

None.

x87 Condition Code

None affected.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction preceded by LOCK prefix. The emulate bit (EM) of CR0 is set to 1. The FXSAVE/FSRSTOR bit (bit 24) in CPUID standard function 1 or extended function 8000_0001 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 is set to 1.
Stack, #SS			X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
General protection, #GP	X	X	X	The memory operand is not aligned on a 16-byte boundary.
General protection, #GP			X	During instruction execution, the effective address of one of the operands points to an illegal memory location (except that illegal stack-segment memory references are reported by #SS). An attempt is made to write a non-zero value to reserved bits in MXCSR.
General protection, segment overrun, #GP	X	X		The address of a data operand is outside the address range 0000h to FFFFh.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	If alignment checks are enabled (i.e., the AC bit of rFLAGS is set to 1, the AM bit of CR0 are set to 1, and in protected mode, CPL = 3), reporting of #AC vs. #GP is a function of the misalignment and is implementation-dependent. If alignment checks are not enabled, only 16-byte misalignment is reported via #GP.

FXTRACT Floating-Point Extract Exponent and Significand

Takes the exponent portion from the floating-point value stored in ST(0), and places it into ST(1). Extracts the significand from ST(0) and places it in ST(0).

The sign and significand of this value are the same as those found in the source operand, and the exponent is 3FFFH (biased value for a true exponent of zero). The ST(1) register contains the value of the original operand's true (unbiased) exponent expressed as a floating-point number.

This instruction and the F2XM1 instruction are useful for performing scaling operations. The FXTRACT instruction is also useful for converting numbers in double-extended-precision format to decimal representations (e.g., for printing or displaying).

If the floating-point zero-divide exception (#Z) is masked and the source operand is zero, an exponent value of $-\infty$ is stored in register ST(1) and 0 with the sign of the source operand is stored in register ST(0).

Mnemonic	Opcode	Description
FXTRACT	D9 F4	Extract the exponent and significand of ST(0) and put the exponent in ST(0) and push the significand onto the floating-point register stack.

Related Instructions

FABS, FPREM, FRNDINT, FCHS

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	If no other flags are set.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	The source operand is an SNaN value or unsupported format.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	ST(0) operand is ± 0 .

FYL2X**Floating-Point $y * \log_2 x$**

Computes $z = ST(1) * \log_2 ST(0)$. $ST(0)$ must be greater than 0. Z replaces $ST(1)$, which becomes the new stack top after $ST(0)$ is popped off.

Mnemonic	Opcode	Description
FYL2X	D9 F1	$ST(1)$ becomes $ST(1) * \log_2 ST(0)$ with pop of the floating-point register stack.

If the divide-by-zero exception is masked and register $ST(0)$ contains ± 0 , the instruction returns with a sign that is the opposite of the sign of the source operand in register $ST(1)$.

The FYL2X instruction is designed with a built-in multiplication to optimize the calculation of logarithms with an arbitrary positive base (b):

$$\log_x y = \log_2 y / \log_2 x$$

Using the fact that $x^y = 2^{y \log_2 x}$ this instruction can be used to calculate exponential functions.

Related Instructions

FYL2XP1, F2XM1

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Description
C0	Undefined
C1	0 indicates a stack underflow, if both invalid operation bit (4) and stack fault bit (6) are set.
	1 indicates a stack overflow, if both invalid operation bit (4) and stack fault bit (6) are set.
	0 indicates no round up, if inexact result bit (5) is 1.
	1 indicates round up, if inexact result bit (5) is 1.
C2	Undefined

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Either operand is a SNaN value or unsupported formats.
	X	X	X	Source operand in register ST(0) is a negative finite value (not -0).
	X	X	X	Source operand in register ST(0) is ± 0 .
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The source operand is a denormal value.
Zero-divide exception (ZE)	X	X	X	ST(0) operand is ± 0 .
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

FYL2XP1 Floating-Point $y * \log_2(X+1)$

The FYL2XP1 computes $Z = Y \bullet \log_2(X + 1)$. X is from ST(0) and must be in the range: $0 < |X| < (1 - 2^{1/2}) / 2$. Y is from ST(1). Z replaces Y, which becomes the new stack top as X is popped off.

Mnemonic	Opcode	Description
FYL2XP1	D9 F9	ST(1) becomes $ST(1) * \log_2(ST(0) + 1.0)$ with pop of the floating-point register stack.

Related Instructions

FYL2X, F2XM1

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	0	Stack underflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	1	Stack overflow, if both invalid operation bit (4) and stack fault bit (6) are 1.
	0	No round up, if inexact result bit (5) is 1.
	1	Round up, if inexact result bit (5) is 1.
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	The emulate bit (EM) or the task switch bit (TS) of the control register (CR0) is set to 1.
x87 Floating-Point Exception Pending, #MF				
Invalid-operation exception (IE)	X	X	X	Either operand is an SNaN values or unsupported formats.
Invalid-operation exception (IE) with stack fault (SF)	X	X	X	A stack overflow or underflow occurred.
Denormalized-oper- and exception (DE)	X	X	X	The instruction attempts to operate on a denormal operand. An attempt is made to load a denormal single-precision or double-precision floating-point value into an x87 register.
Zero-divide exception (ZE)	X	X	X	The source operand is a denormal value.
Overflow exception (OE)	X	X	X	The rounded result is too large to fit into the floating-point format of the destination operand.
Underflow exception (UE)	X	X	X	The rounded result is too small to fit into the floating-point format of the destination operand.
Precision exception (PE)	X	X	X	The result cannot be represented exactly in the destination format.

WAIT**Wait for Unmasked x87 Floating-Point Exceptions**

Causes the processor to check for and handle pending, unmasked, floating-point exceptions before proceeding. (FWAIT is an alternate mnemonic for the WAIT). Exception-condition reporting (i.e., assertion of the FERR# output signal) occurs immediately when an error occurs, although the servicing of such an exception occurs when the next x87 non-control instruction is executed (if CR0.NE=1) or when a later instruction, depending on chip-set delay, is executed (if CR0.NE=0).

This instruction is useful for synchronizing exceptions in critical sections of code. Coding a WAIT instruction after a floating-point instruction insures that any unmasked floating-point exceptions are handled before the processor can modify the instruction results.

Mnemonic	Opcode	Description
WAIT	9B	Check for any outstanding x87 floating-point exceptions.

Related Instructions

FWAIT

rFLAGS Affected

None.

x87 Condition Code

x87 Condition Code	Value	Description
C0	U	
C1	U	
C2	U	
C3	U	
<i>U indicates 'undefined.'</i>		

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM	X	X	X	MP and TS in CR0 are set.

2.10 System Instruction Reference

This section describes the function, mnemonic syntax, opcodes, affected flags, and possible exceptions generated by the system instructions. The system instructions are used to establish the operating mode, access processor resources, handle program and system errors, and manage memory. Many of these instructions can only be executed by software, such as the operating system kernel and interrupt handlers, which runs at the highest privilege level. Only system instructions can access certain processor resources, such as the control registers, model-specific registers, and debug registers.

System instructions are supported in all hardware implementations of the x86-64 architecture, except that the following system instructions are implemented only if their associated CPUID function bit is set:

- SYSENTER and SYSEXIT, indicated by bit 11 of CPUID standard function 1.
- SYSCALL and SYSRET, indicated by bit 11 of CPUID extended function 8000_0001h.
- Long Mode instructions, indicated by bit 29 of CPUID extended function 8000_0001h.
- RDMSR and WRMSR, indicated by bit 5 of CPUID standard function 1 or extended function 8000_0001h.

There are also several other CPUID function bits that control the use of system resources and functions, such as paging functions, virtual-mode extensions, machine-check exceptions, advanced programmable interrupt control (APIC), memory-type range registers (MTRRs), etc. For details, see Section 3.3, “Processor Feature Identification”, on page 77 of Volume 2.

For further information about the system instructions and register resources, see:

- Volume 2: Section 6, “System-Management Instructions”, on page 189.
- Volume 3: Section 2.3, “Summary of Registers and Data Types”, on page 28.
- Volume 3: Section 2.5, “Notation”, on page 41.
- Volume 3: Section 1.2, “Instruction Prefixes”, on page 3.

ARPL**Adjust Requestor Privilege Level**

The ARPL instruction compares the requestor privilege level (RPL) fields of two segment selectors in the source and destination operands of the instruction. The destination operand can be either a 16-bit register or memory location; the source operand must be a 16-bit register. If the RPL field of the destination operand is less than the RPL field of the segment selector in the source register, then the zero flag is set and the RPL field of the destination operand is increased to match that of the source operand. Otherwise, the destination operand remains unchanged and the zero flag is cleared.

Mnemonic	Opcode	Description
ARPL <i>reg/mem16</i>	63/r	Adjust the requested privilege level of a destination 16-bit register or memory operand to a level not less than the requested privilege level of the segment selector specified in the 16-bit source register. (Invalid in 64-bit mode.)

The ARPL instruction is intended for use by operating-system procedures to adjust the RPL of a segment selector that has been passed to the operating system by an application program to match the privilege level of the application program. The segment selector passed to the operating system is placed in the destination operand and the segment selector for the code segment of the application program is placed in the source operand. The RPL field in the source operand represents the privilege level of the application program. The ARPL instruction then insures that the RPL of the segment selector received by the operating system is no lower than the privilege level of the application program.

See Section 6.4.2, “Adjusting Access Rights”, on page 200 of Volume 2, for more information access rights.

In 64-bit mode, this opcode (63H) is used for the MOVSSD instruction.

Related Instructions

LAR, LSL, VERR

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
													1/0			
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
			X	Result is located in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

CLI Clear Interrupt Flag

Resets the interrupt flag (IF) in the rFLAGS register to zero, thereby masking external interrupts received on INTR input. Interrupts received on non-maskable interrupt (NMI) input are not blocked by this instruction.

Mnemonic	Opcode	Description
CLI	FA	Clear the interrupt flag to zero.

In protected mode and virtual-8086-mode, this instruction is IOPL-sensitive. The processor takes a general-protection exception (#GP) whenever an IOPL-sensitive instruction is executed and the EFLAGS.IOPL field is less than the CPL. Because all virtual-8086 programs run at CPL=3, system software can interrupt all instructions that modify the IF bit by setting the IOPL less than 3.

In virtual-8086 mode, if the virtual-8086-mode extensions are enabled (CR4.VME = 1) or if the protected mode virtual interrupt (PVI) flag is set in protected mode (CR4.VPI = 1), then the CLI instruction also clears the rFlags.VIF flag.

See Section 8.9.1, “Virtual-8086 Mode Extensions”, on page 308 of Volume 2 for more information about IOPL-sensitive instructions.

Action

```
if (CPL <= IOPL)
    EFLAGS.IOPL = 0;

else if ((mode==virtual) && CR4.VME) || ((mode==protected)&&CR4.PVI) && (CPL ==3)
    EFLAGS.VIF = 0;

else
    #GP
```

Related Instructions

STI.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
		0								0						
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is greater than the I/O privilege level of the current task.

CLTS**Clear Task-Switched Flag in CR0**

Clears the task-switched (TS) flag in the CR0 register to 0. The processor sets the TS flag on each task switch. The CLTS instruction is intended to facilitate the synchronization of FPU context saves during multitasking operations.

Mnemonic	Opcode	Description
CLTS	0F 06	Clears the task-switched flag in control register 0.

This instruction can only be used if the current privilege level is 0.

See Section 3.1, “System-Control Registers”, on page 53 of Volume 2 for more information on FPU synchronization and the TS flag.

Action

```
taskSwitchedFlag = 0;
```

Related Instructions

MOV CR_n

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is greater than 0.

HLT**Halt**

Causes the microprocessor to halt instruction execution and enter the HALT state. Execution resumes when an unmasked hardware interrupt (INTR), non-maskable interrupt (NMI), system management interrupt (SMI), RESET or INIT occurs. Entering the halt state puts the processor in low-power mode.

Mnemonic	Opcode	Description
HLT	F4	Halts instruction execution.

If an INTR or NMI is used to resume execution after a HLT instruction, the saved instruction pointer points to the instruction following the HLT instruction.

Before executing a HLT instruction, you should enable hardware interrupts (with the STI instruction). Do not clear the interrupt flag (IF) with the CLI instruction before executing a HLT instruction. Doing so disables INTRs, and the system will remain in a halt state until an NMI, SMI, RESET or INIT occurs.

If an SMI brings the processor out of the halt state, the SMI handler can decide whether to return to the HALT state or not. See Volume 2: System Programming for information on SMIs.

Program privilege level must be 0 to execute this instruction.

Related Instructions

STI, CLI.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is not 0.

INT 3 Interrupt to Debug Vector

Calls the debug exception handler. This is the one-byte version of the two-byte INT 3 instruction described on page 140. This one-byte INT 3 instruction differs from the two-byte INT 3 instruction in the following ways:

- The INT 3 interrupt is handled without any IOPL checking.
- In VME mode, the interrupt is not redirected via the interrupt redirection table.

Mnemonic	Opcode	Description
INT 3	CC	Trap to debugger at Interrupt 3.

For complete descriptions of the steps performed by INT instructions, see the following:

- *Legacy-Mode Interrupts*: Section 8.7, “Legacy-Mode Interrupt Control Transfers”, on page 285 of Volume 2.
- *Long-Mode Interrupts*: Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

INT3_START:

```
IF (REAL_MODE)
    INT3_REAL
ELIF (PROTECTED_MODE)
    INT3_PROTECTED
ELSE // (VIRTUAL_MODE)
    INT3_VIRTUAL
```

INT3_REAL:

```
temp_RIP = READ_MEM.w [idt:3*4]
           // read target CS:RIP from the real-mode idt, entry 3
temp_CS  = READ_MEM.w [idt:3*4+2]

PUSH.w old_RFLAGS
PUSH.w old_CS
PUSH.w old_RIP

IF (temp_RIP>CS.limit)
    EXCEPTION [#GP]
```

```

CS.sel  = temp_CS
CS.base = temp_CS SHL 4

RFLAGS.AC,TF,IF,RF cleared
RIP = temp_RIP
EXIT

```

INT3_PROTECTED:

```

temp_vector = 3    // go to int-3 handler
temp_idt_desc = READ_IDT (temp_vector)

IF (temp_idt_desc.attr.type = 'taskgate')
    TASK_SWITCH    // using tss selector in the task gate as the target tss

temp_RIP = temp_idt_desc.offset

IF (LONG_MODE)    // in long mode, we need to read the 2nd half of a 16-byte
    interrupt-gate
{
    // from the gdt/ldt, to get the upper 32 bits of the
    // target RIP
    temp_upper = READ_MEM.q [idt:3*16+8]
    IF (temp_upper's extended attribute bits != 0)
        EXCEPTION [#GP(0)]    // the extended attribute bits must be all zero
    temp_RIP = temp_RIP + (temp_upper SHL 32) // concatenate both halves of RIP
}

IF (LONG_MODE)    // the size of the gate controls the size of the stack pushes
    V=8-byte      // long mode only uses 64-bit gates
ELIF ((temp_idt_desc.attr.type = 'intgate32')
    || (temp_idt_desc.attr.type = 'trapgate32'))
    V=4-byte      // legacy mode, using a 32-bit gate
ELSE // gate is intgate16 or trapgate16
    V=2-byte      // legacy mode, using a 16-bit gate

CS = READ_DESCRIPTOR (temp_idt_desc.segment, intcs_chk)

temp_CPL = CS.sel.rpl

IF (CPL=temp_CPL)
{
    IF (LONG_MODE)
    {
        IF (temp_idt_desc.ist!=0)
            // in long mode, if the idt gate specifies an ist pointer,
            // a stack-switch is always done
            RSP = READ_MEM.q [tss:ist_index*8+28]
            RSP = RSP AND 0xFFFFFFFFFFFFFFF0
            // in long mode, interrupts/exceptions align rsp to a
            // 16-byte boundary

```

```

        PUSH.q old_SS
            // in long mode, SS:RSP is always pushed to the stack
        PUSH.q old_RSP
    }

    PUSH.v old_RFLAGS
    PUSH.v old_CS
    PUSH.v old_RIP

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
    {
        EXCEPTION [#GP(0)]
    }

    RFLAGS.VM,NT,TF,RF cleared
    RFLAGS.IF cleared if interrupt gate

    RIP = temp_RIP
    EXIT
}
ELSE // (CPL > temp_CPL)
{
    CPL = temp_CPL

    temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER (CPL,
        temp_idt_desc.ist)

    IF (LONG_MODE)
        temp_RSP = temp_RSP AND 0xFFFFFFFFFFFFFFF0
            // in long mode, interrupts/exceptions round rsp to a
            // 16-byte boundary

    RSP.q = temp_RSP
    SS = temp_SS_desc

    PUSH.v old_SS
    PUSH.v old_RSP
    PUSH.v old_RFLAGS
    PUSH.v old_CS
    PUSH.v old_RIP

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
    {
        EXCEPTION [#GP(CS.sel)]
    }

    RFLAGS.VM,NT,TF,RF cleared
    RFLAGS.IF cleared if interrupt gate

```



```

        RIP = temp_RIP
        EXIT
    }

```

INT3_VIRTUAL:

```

temp_vector = 3                // go to int-3 handler
temp_idt_desc = READ_IDT (temp_vector)

IF (temp_idt_desc.attr.type = 'taskgate')
    TASK_SWITCH                // using tss selector in the task gate as the
                                // target tss

IF ((temp_idt_desc.attr.type = 'intgate32')
    || (temp_idt_desc.attr.type = 'trapgate32'))
    // the size of the gate controls the size of the
    // stack pushes
{
    V=4-byte                    // legacy mode, using a 32-bit gate
}
ELSE // gate is intgate16 or trapgate16
    V=2-byte                    // legacy mode, using a 16-bit gate

temp_RIP = temp_idt_desc.offset
CS = READ_DESCRIPTOR (temp_idt_desc.sel, intcs_chk)

IF (CS.sel.rpl!=0)             // handler must run at cpl 0
    EXCEPTION [#GP(0)]

CPL = 0

temp_ist = 0                   // call-far doesn't use ist pointers
temp_SS_desc:temp_RSP = READ_INNER_LEVEL_STACK_POINTER (CPL, temp_ist)

RSP.q = temp_RSP
SS = temp_SS_desc

PUSH.v old_GS
PUSH.v old_FS
PUSH.v old_DS
PUSH.v old_ES
PUSH.v old_SS
PUSH.v old_RSP
PUSH.v old_RFLAGS             // pushed with RF clear
PUSH.v old_CS
PUSH.v old_RIP

IF (temp_RIP > CS.limit)
    EXCEPTION [#GP(CS.sel)]

```

```

DS = NULL    // can't use virtual-mode selectors in protected mode
ES = NULL    // can't use virtual-mode selectors in protected mode
FS = NULL    // can't use virtual-mode selectors in protected mode
GS = NULL    // can't use virtual-mode selectors in protected mode

```

```

RFLAGS.VM,NT,TF,RF cleared
RFLAGS.IF cleared if interrupt gate

```

```

RIP = temp_RIP
EXIT

```

Related Instructions

INT, INTO, IRET.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
			1/0	1/0	1/0	1/0				1/0	1/0					
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Breakpoint, #BP	X	X	X	INT 3 instruction is executed.
Invalid TSS, #TS (selector)		X	X	The TSS's stack-segment selector is null.
		X	X	The new stack-segment selector in the TSS contains an RPL that is not equal to the DPL of the code-segment descriptor that is being accessed by the interrupt or trap gate by the interrupt or trap gate.
		X	X	The target stack-segment descriptor contains a DPL that is not equal to that of the code-segment descriptor being accessed by the interrupt or trap gate.
		X	X	The TSS's stack-segment is not a writable data segment.
Segment not present, #NP (selector)		X	X	The index in the stack-segment selector is beyond the descriptor table's limits.
		X	X	One of the following is not present: code segment, TSS, trap gate, task gate, or interrupt gate.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X		X	During instruction execution, the effective address of a stack operand points to an illegal stack-memory location. (Illegal references to non-stack memory are reported by #GP.)
Stack, #SS (selector)		X	X	The stack segment limit was exceeded when a stack switch occurred.
			X	The target stack segment is not present.
General protection, #GP		X	X	The instruction pointer in the in the task gate, interrupt gate, trap gate, or IDT is outside the limits of the code segment.
	X			During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
	X			The exception vector is beyond the limits of the IDT.
General protection, #GP (selector)		X		If the IOPL is less than 3 and the DPL of the interrupt-, trap-, or task-gate descriptor is not equal to 3.
		X	X	The segment selector is null in the trap, interrupt, or task gate.
		X	X	The selector index for the TSS, gate, or code segment, is beyond the limits of the descriptor table.
		X	X	The exception vector is beyond the limits of the IDT.
		X	X	The descriptor in the IDT is not for an interrupt, trap, or task gate.
		X	X	An interrupt is generated by the INTn instruction and the DPL of an interrupt, trap, or task gate is less than the CPL. (For protected mode, also INT 3 or INTO instruction)
		X	X	The segment selector in a trap gate or interrupt does not point to a code-segment descriptor.
		X	X	The TI bit in the TSS's segment selector indicates a local table.
Page fault, #PF		X	X	The segment descriptor for the TSS indicates that the TSS is not available or busy.
		X	X	A page fault resulted from the execution of the instruction.

INVD Invalidate Caches

The INVD instruction flushes internal caches (data cache, instruction cache, and on-chip L2 cache) and triggers a bus cycle that causes external caches to flush themselves as well. This instruction does not invalidate TLB caches.

Mnemonic	Opcode	Description
INVD	0F 08	Flush internal caches and trigger external cache flushes.

No data is written back to main memory from flushed internal caches. After flushing internal caches, the processor proceeds immediately with the execution of the next instruction without waiting for external hardware to flush its caches.

This is a privileged instruction. The current privilege level (CPL) of a procedure invalidating the processor's internal caches must be zero.

To insure that data is written back to memory prior to invalidating caches, use the WBINVD instruction.

The INVD instruction is a serializing instruction.

Related Instructions

WBINVD

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is not 0.

INVLPG Invalidate TLB Entry

The INVLPG instruction takes a 1-byte operand and invalidates the TLB entry that would be used for that byte.

Mnemonic	Opcode	Description
INVLPG <i>mem8</i>	0F 01 /7	Invalidate the TLB for the page containing a specified memory location.

This instruction invalidates the TLB entry, regardless of the G (Global) bit setting in the associated PDE or PTE entry and regardless of the the page size (4K, 2M or 4M). It may invalidate any number of additional TLB entries, in addition to the targeted entry.

INVLPG is a serializing instruction and a privileged instruction. The current privilege level must be zero to execute this instruction in legacy protected mode or long mode.

See Section 5, “Page Translation and Protection”, on page 155 of Volume 2 for more information on page translation.

Related Instructions

MOV CR*n* (CR3 and CR4), SWAPGS.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Operand is a register.
General protection, #GP		X	X	The current privilege level is not 0.


```

    IRET_PROTECTED
ELSE // (VIRTUAL_MODE)
    IRET_VIRTUAL

```

```
IRET_REAL:
```

```

    POP.v temp_RIP
    POP.v temp_CS
    POP.v temp_RFLAGS

    IF (temp_RIP > CS.limit)
        EXCEPTION [#GP]

    CS.sel = temp_CS
    CS.base = temp_CS SHL 4

    RFLAGS.v = temp_RFLAGS // VIF,VIP,VM unchanged
    RIP = temp_RIP
    EXIT

```

```
IRET_PROTECTED:
```

```

    IF (RFLAGS.NT=1)                // iret does a task-switch to a previous task
        IF (LEGACY_MODE)
            TASK_SWITCH              // using the 'back link' field in the tss
        ELSE
            EXCEPTION [#GP(0)]      // (LONG_MODE)
            // task switches aren't supported in long mode

    POP.v temp_RIP
    POP.v temp_CS
    POP.v temp_RFLAGS

    IF ((temp_RFLAGS.VM=1) && (CPL=0) && (LEGACY_MODE))
        IRET_FROM_PROTECTED_TO_VIRTUAL

    temp_CPL = temp_CS.rpl

    IF ((64BIT_MODE) || (temp_CPL!=CPL))
    {
        POP.v temp_RSP              // in 64-bit mode, iret always pops ss:rsp
        POP.v temp_SS
    }

    CS = READ_DESCRIPTOR (temp_CS, iret_chk)

    IF ((64BIT_MODE) && (temp_RIP is non-canonical)
        || (!64BIT_MODE) && (temp_RIP > CS.limit))
    {
        EXCEPTION [#GP(0)]
    }

```

```

}

CPL = temp_CPL

IF ((started in 64-bit mode) || (changing cpl))
    // ss:rsp were popped, so load them into the registers
{
    SS = READ_DESCRIPTOR (temp_SS, ss_chk)
    RSP.s = temp_RSP
}

FOR (seg = ES, DS, FS, GS)
    IF ((seg.attr.dpl < CPL) && ((seg.attr.type = 'data')
        || (seg.attr.type = 'non-conforming-code')))
    {
        seg = NULL        // can't use lower dpl data segment at higher cpl
    }

RFLAGS.v = temp_RFLAGS    // VIF,VIP,IOPL only changed if (old_CPL=0)
                        // IF only changed if (old_CPL<=old_RFLAGS.IOPL)
                        // VM unchanged
                        // RF cleared

RIP = temp_RIP
EXIT

```

IRET_VIRTUAL:

```

IF ((RFLAGS.IOPL<3) && (CR4.VME=0))
    EXCEPTION [#GP(0)]

POP.v temp_RIP
POP.v temp_CS
POP.v temp_RFLAGS

IF (temp_RIP > CS.limit)
    EXCEPTION [#GP(0)]

IF (RFLAGS.IOPL=3)
{
    RFLAGS.v = temp_RFLAGS    // VIF,VIP,VM,IOPL unchanged
                        // RF cleared

    CS.sel = temp_CS
    CS.base = temp_CS SHL 4

    RIP = temp_RIP
    EXIT
}

// now ((IOPL<3) && (CR4.VME=1))

```



```

    ELSIF ((OPERAND_SIZE=16) && !((temp_RFLAGS.IF=1)
        && (RFLAGS.VIP=1)) && (temp_RFLAGS.TF=0))
    {
        RFLAGS.w = temp_RFLAGS // RFLAGS.VIF=temp_rFLAGS.IF
                                // IF,VIP,VM,IOPL unchanged
                                // RF cleared

        CS.sel = temp_CS
        CS.base = temp_CS SHL 4

        RIP = temp_RIP
        EXIT
    }
    ELSE // ((RFLAGS.IOPL<3) && (CR4.VME=1) && ((OPERAND_SIZE=32) ||
        ((temp_RFLAGS.IF=1) && (RFLAGS.VIP=1)) || (temp_RFLAGS.TF=1)))
        EXCEPTION [#GP(0)]

```

IRET_FROM_PROTECTED_TO_VIRTUAL:

```

    // temp_RIP already popped
    // temp_CS already popped
    // temp_RFLAGS already popped, temp_RFLAGS.VM=1

    POP.d temp_RSP
    POP.d temp_SS
    POP.d temp_ES
    POP.d temp_DS
    POP.d temp_FS
    POP.d temp_GS

    CS.sel = temp_CS // force the segments to have virtual-mode values
    CS.base = temp_CS SHL 4
    CS.limit= 0x0000FFFF
    CS.attr = 16-bit dpl3 code

    SS.sel = temp_SS
    SS.base = temp_SS SHL 4
    SS.limit= 0x0000FFFF
    SS.attr = 16-bit dpl3 stack

    DS.sel = temp_DS
    DS.base = temp_DS SHL 4
    DS.limit= 0x0000FFFF
    DS.attr = 16-bit dpl3 data

    ES.sel = temp_ES
    ES.base = temp_ES SHL 4
    ES.limit= 0x0000FFFF
    ES.attr = 16-bit dpl3 data

    FS.sel = temp_FS

```

```

FS.base = temp_FS SHL 4
FS.limit= 0x0000FFFF
FS.attr = 16-bit dpl3 data

GS.sel  = temp_GS
GS.base = temp_GS SHL 4
GS.limit= 0x0000FFFF
GS.attr = 16-bit dpl3 data

RSP.d = temp_RSP
RFLAGS.d = temp_RFLAGS
CPL = 3

RIP = temp_RIP AND 0x0000FFFF
EXIT

```

Related Instructions

INT, INTO, INT3, RET, IRETD, IRETQ.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Segment not present, #NP (selector)			X	The return code or stack segment is not present.
Stack, #SS	X	X	X	The top bytes of the stack are not within stack limits.
General protection, #GP		X	X	The return instruction pointer is not within the return code segment limit.
			X	The IOPL is not equal to 3.
			X	The return code or stack segment selector is null.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP (selector)			X	The segment selector index is outside the descriptor table limits.
			X	The return code segment selector RPL is greater than the CPL.
			X	The DPL for a nonconforming-code segment is not equal to the RPL of the code segment selector.
			X	The DPL for a conforming-code segment is greater than the return code segment selector RPL.
			X	The stack segment descriptor DPL is not equal to the RPL of the return code segment selector.
			X	The stack segment is not a writable data segment.
			X	The stack segment selector RPL is not equal to the RPL of the return code segment selector.
			X	The segment descriptor for a code segment selector does not indicate it is a code segment.
			X	The TI bit in the TSS's segment selector indicates a local table.
			X	The segment descriptor for the TSS indicates that the TSS is not available or busy.
General protection, segment overrun, #GP	X	X		The return instruction pointer is not within the return code segment limit.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment check bit (AC) of the rFLAGS register and the alignment mask bit (AM) of CR0 are set to 1. (In protected mode, CPL = 3.)

LAR Load Access Rights Byte

The LAR instruction loads the access rights from the segment descriptor specified by a 16-bit, 32-bit, or 64-bit source register or memory operand into a specified 16-bit general-purpose register and sets the zero (ZF) flag in the rFLAGS register if successful.

Mnemonic	Opcode	Description
LAR <i>reg16, reg/mem16</i>	0F 02 /r	Reads the GDT/LDT descriptor referenced by the 16-bit source operand, masks the attributes with FF00h and saves the result in the 16-bit destination register.
LAR <i>reg32, reg/mem32</i>	0F 02 /r	Reads the GDT/LDT descriptor referenced by the 32-bit source operand, masks the attributes with 00FFFF00h and saves the result in the 32-bit destination register.
LAR <i>reg64, reg/mem64</i>	0F 02 /r	Reads the GDT/LDT descriptor referenced by the 32-bit source operand, masks the attributes with 00FFFF00h and saves the result in the 64-bit destination register.

The LAR instruction performs the following:

- Checks that the segment selector is not null.
- Check whether the specified segment selector points to a descriptor that is within the limits of the global descriptor table or local descriptor table being accessed.
- Checks whether the specified segment descriptor is visible at the current privilege level (CPL). That is, for segments that are not conforming code segments, the RPL of both the CPL and selector must not be greater than the descriptor's DPL
- Checks that the descriptor type is valid for the LAR instruction. Valid descriptor types include the following:

Valid Descriptor Type	Description
—	All code or data descriptors
1	Available 16-bit TSS
2	LDT
3	Busy 16-bit TSS
4	16-bit call gate
5	16-bit or 32-bit task gate

Valid Descriptor Type	Description
9	Available 32-bit TSS
B	Busy 32-bit TSS
C	32-bit call gate

If the segment selector passes these checks, it is loaded into the destination general-purpose register. If it does not, then the zero flag is cleared and the destination register is not modified.

When the operand size is 16 bits, access rights include the DPL and Type fields located bytes 4 and 5 of the descriptor table entry. Before loading the access rights into the destination operand, the low order word is masked with FF00H.

When the operand size is 32 or 64 bits, access rights include the DPL and type as well as the descriptor type (S field), segment present (P flag), available to system (AVL flag), default operation size (D/B flag), and granularity flags located in bytes 4–7 of the descriptor. Before being loaded into the destination operand, the doubleword is masked with 00FF_FF00H.

In 64-bit mode, for both 32-bit and 64-bit operand sizes, 32-bit register results are zero-extended to 64 bits.

This instruction can only be executed in protected mode.

Related Instructions

ARPL, LSL, VERR, VERW.

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
													1/0			
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or zero is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location or a non-canonical address was generated.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and alignment checking is enabled and EFLAGS.AC=1 and CPL=3.

LGDT**Load Global Descriptor Table Register**

Loads the pseudo-descriptor specified by the source operand into the global descriptor table register (GDTR). The pseudo-descriptor is a memory location containing the GDTR base and limit. In legacy and compatibility mode the pseudo-descriptor is six bytes; in 64-bit mode it is 10 bytes.

Mnemonic	Opcode	Description
LGDT <i>mem16:32</i>	0F 01 /2	Loads <i>mem16:32</i> into the global descriptor table register.
LGDT <i>mem16:64</i>	0F 01 /2	Loads <i>mem16:64</i> into the global descriptor table register.

If the operand size is 16 bits, the high-order byte of the 6-byte operand is not used. The lower two bytes specify the 16-bit limit and the third, fourth, and fifth bytes specify the 24-bit base address. The high-order byte of the GDTR is filled with zeros.

If the operand size is 32 bits, the lower two bytes specify the 16-bit limit and the upper 4 bytes specify a 32-bit base address.

In 64-bit mode, the lower two bytes specify the 16-bit limit and the upper eight bytes specify a 64-bit base address. In 64-bit mode, size prefixes are ignored and therefore the pseudo-descriptor is always 10 bytes.

This instruction is only used in operating system software and must be executed at privilege level 0. It must be executed at least once in real mode to initialize the processor before switching to protected mode.

This is a serializing instruction.

Related Instructions

LIDT, LLDT, LTR, SGDT, SIDT, STR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X		X	The source operand is not a memory location.
Stack, #SS	X		X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP	X	X	X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
			X	The current privilege level is not 0.
			X	The LGDT base address is not canonical.
			X	The segment register used to access memory contains a null selector.
			X	
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

LIDT**Load Interrupt Descriptor Table Register**

Loads the pseudo-descriptor specified by the source operand into the interrupt descriptor table register (IDTR). The pseudo-descriptor is a memory location containing the IDTR base and limit. In legacy and compatibility mode the pseudo-descriptor is six bytes; in 64-bit mode it is 10 bytes.

Mnemonic	Opcode	Description
LIDT <i>mem16:32</i>	0F 01 /3	Loads <i>mem16:32</i> into the interrupt descriptor table register.
LIDT <i>mem16:64</i>	0F 01 /3	Loads <i>mem16:64</i> into the interrupt descriptor table register.

If the operand size is 16 bits, the high-order byte of the 6-byte operand is not used. The lower two bytes specify the 16-bit limit and the third, fourth, and fifth bytes specify the 24-bit base address. The high-order byte of the IDTR is filled with zeros.

If the operand size is 32 bits, the lower two bytes specify the 16-bit limit and the upper 4 bytes specify a 32-bit base address. The high-order byte of the IDTR is filled with zeros.

In 64-bit mode, the lower two bytes specify the 16-bit limit and the upper eight bytes specify a 64-bit base address. In 64-bit mode, size prefixes are ignored and the operand size is forced to 64 bits, therefore the pseudo-descriptor is always 10 bytes.

This instruction is only used in operating system software and must be executed at privilege level 0. It is normally executed at least once in real mode to initialize the processor before switching to protected mode.

This is a serializing instruction.

Related Instructions

LGDT, LLDT, LTR, SGDT, SIDT, STR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X		X	The source operand is not a memory location.
Stack, #SS	X		X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP	X	X	X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
		X	X	The current privilege level is not 0.
			X	The segment register used to access memory contains a null selector.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

LLDT**Load Local Descriptor Table Register**

Loads the descriptor pointed to by the specified segment selector into the local descriptor table register (LDTR). The processor uses the selector to locate the descriptor for the LDT in the global descriptor table. It then loads this descriptor into the LDTR.

Mnemonic	Opcode	Description
LLDT <i>reg/mem16</i>	0F 00 /2	Loads the segment select at a specified 16-bit memory location into the local descriptor table register.

If the source operand is null, the LDTR is marked invalid and all references to descriptors in the LDT will generate a general protection exception (#GP), except for the LAR, VERR, VERW or LSL instructions.

In 64-bit mode, the instruction references a 16-byte descriptor that contains a 64-bit base. The LDT type (02H) is redefined in 64-bit mode for use as the 16-byte LDT descriptor.

This instruction must be executed in protected mode. It is only provided for use by operating system software at CPL 0.

This is a serializing instruction.

Related Instructions

LGDT, LIDT, LTR, SGDT, SIDT, SLDT, STR.

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Segment not present, #NP (selector)			X	The LDT descriptor is not present.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
			X	The current privilege level is not 0.
			X	The segment register used to access memory contains a null selector.
General protection, #GP (selector)			X	The selector operand does not point into the Global Descriptor Table or if the entry in the GDT is not Local Descriptor Table.
			X	Segment selector is beyond GDT limit.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

LMSW**Load Machine Status Word**

The LMSW instruction loads the contents of the 16-bit register or memory operand into bits 0 through 15 of the machine status word in register CR0. Only the protection enabled (PE), monitor coprocessor (MP), emulation (EM), and task switched (TS) bits of CR0 are modified. Additionally, LMSW can set CR0.PE, but cannot clear it.

Mnemonic	Opcode	Description
LMSW <i>reg/mem16</i>	0F 01 /6	Loads the lower four bits of the source into the lower 4 bits of CR0.

This instruction cannot be used in legacy real mode. The LMSW instruction can only be used when the current privilege level is zero. It is only provided for compatibility with early processors and should only be used in operating system software.

Use the MOV instruction to load all 32 bits of CR0.

Related Instructions

MOV, SMSW.

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP		X	X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
			X	The current privilege level is not 0.
			X	The segment register used to access memory contains a null selector.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

LTR Load Task Register

Loads the task state segment (TSS) descriptor pointed to by the specified segment selector into the task register (TR). The processor uses the selector to locate the descriptor for the TSS in the global descriptor table. It then loads this descriptor into the TR. The TSS is marked as busy, but no task switch is made.

Mnemonic	Opcode	Description
LTR <i>reg/mem16</i>	0F 00 /3	Load the contents of a 16-bit register or memory operand into the task register.

If the source operand is null, a general protection exception (#GP) is generated.

In 64-bit mode, the instruction references a 64-bit mode descriptor to load a 64-bit base. The TSS type (09H) is redefined in 64-bit mode for use as the 16-byte TSS descriptor.

This instruction must be executed in protected mode when the current privilege level is zero. It is only provided for use by operating system software.

The operand size attribute has no effect on this instruction.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		The instruction is not recognized.
Segment not present, #NP (selector)			X	The TSS is marked not present.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
		X	X	The current privilege level is not 0.
			X	The segment register used to access memory contains a null selector.
General protection, #GP (selector)			X	The source selector points to a segment that is not a TSS or to one for a task that is already busy.
			X	The selector points to LDT or is beyond GDT limit.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

MOV(CR_n)**Move to/from Control Registers**

Moves the contents of a 32-bit or 64-bit general-purpose register to a control register or vice versa.

In 64-bit mode, with the exception of CR8, the operand size is fixed at 64 bits without the need for a REX prefix. In non-64-bit mode, the operand size is fixed at 32 bits and the upper 32 bits is written with zeros for writes.

Mnemonic	Opcode	Description
MOV CR _n , reg32	0F 22 /r	Move the contents of a 32-bit register to CR _n
MOV CR _n , reg64	0F 22 /r	Move the contents of a 64-bit register to CR _n
MOV reg32, CR _n	0F 20 /r	Move the contents of CR _n to a 32-bit register.
MOV reg64, CR _n	0F 20 /r	Move the contents of CR _n to a 64-bit register.

Note:

CR0, CR2, CR3, CR4, and CR8 are the only registers to which this instruction applies. See text for details.

Reading or writing to CR8 requires a REX prefix, and thus can only be accessed in 64-bit mode. However, it can also be read and modified using the task priority register described in Section 3.1, “System-Control Registers”, on page 53 of Volume 2.

CR0 maintains the state of various control bits. CR2 and CR3 are used for page translation. CR4 holds various feature enable bits. CR8 is used to prioritize external interrupts. CR1, CR5, CR6, CR7, and CR9 through CR15 are all reserved and raise an undefined opcode exception (#UD) if referenced.

This instruction is always treated as a register-to-register (MOD = 11) instruction regardless of the encoding of the MOD field in the MODR/M byte.

This is a serializing instruction.

MOV(CR_n) is a privileged instruction and must always be executed at CPL = 0.

Related Instructions

MOV

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is not 0.
			X	An attempt is made to write reserved bits in the page-directory pointers table used in the extended physical addressing mode when the PAE flag in CR4 and the PG flag in control register CR0 are set to 1.
	X		X	An attempt is made to write 1 to any reserved bit in CR4.
			X	An attempt is made to set CR0.PG when CR0.PE = 0.

MOV (DRn)**Move to/from Debug Registers**

Debug registers are 64-bit registers. This instruction moves the contents of a debug register into a 32-bit or 64-bit general-purpose register or vice versa.

Mnemonic	Opcode	Description
MOV <i>reg32</i> , DR0-DR7	0F 21 /r	Move the contents of a debug register to a 32-bit register.
MOV <i>reg64</i> , DR0-DR7	0F 21 /r	Move the contents of a debug register to a 64-bit register.
MOV DR0-DR7, <i>reg32</i>	0F 23 /r	Move the contents of a 32-bit register to a debug register.
MOV DR0-DR7, <i>reg64</i>	0F 23 /r	Move the contents of a 64-bit register to a debug register.

In non-64-bit mode, the operand size is fixed at 32-bits. Writes set the upper halves of the debug register to 0; reads only return the lower 32 bits. In 64-bit mode, the operand size is fixed at 64 bits without the need for a REX prefix.

DR8 through DR15 are reserved and generate an undefined opcode exception if referenced.

This is a serializing instruction. MOV(DR) is also a privileged instruction and must be executed at CPL 0.

The MOV(DR) instruction is always treated as a register-to-register (MOD = 11) instruction regardless of the encoding of the MOD field in the MODR/M byte.

See Section 13, “Debug and Performance Resources”, on page 395 of Volume 2 for details.

Related Instructions

MOV.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Debug, #DB	X		X	Any debug register is accessed while the GD flag in debug register DR7 is set.
Invalid opcode, #UD	X	X	X	The debug extensions (DE) bit in CR4 is set and a MOV instruction is executed involving DR4 or DR5. One of DR8-DR15 was referenced.
General protection, #GP			X	A 1 was written to any of the upper 32 bits of DR6 or DR7 in 64-bit mode..
		X	X	The current privilege level is not 0.

RDMSR**Read Model-Specific Register**

The RDMSR instruction loads the contents of a 64-bit model-specific register (MSR) specified in the ECX register into registers EDX:EAX. The EDX register receives the high-order 32 bits and the EAX register receives the low order bits. The RDMSR instruction is a 32-bit instruction, regardless of operand size; ECX is always holds the MSR number and EDX:EAX holds the data. In 64-bit mode, register results are zero-extended to 64 bits. If a model-specific the register has fewer than 64 bits, the unimplemented bit positions loaded into the destination register are undefined.

Mnemonic	Opcode	Description
RDMSR	0F 32	Copy MSR specified by rCX into rDX:rAX.

This instruction must be executed at a privilege level of 0 or a general protection fault #GP(0) will be raised. This exception is also generated if an attempt is made to specify a reserved or unimplemented model-specific register in ECX.

Use the CPUID instruction to determine if this instruction is supported.

This is a serializing instruction.

For more information about model-specific registers, see Volume 2 and the documentation for various hardware implementations.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X		X	The value in ECX specifies a reserved or unimplemented MSR address.
		X	X	The current privilege level is not 0.

RDPMC**Read Performance-Monitoring Counter**

The RDPMC instruction loads the contents of the 64-bit PerfCtrn register specified by the ECX register (which contains the MSR number for the performance counter), into the EDX and EAX registers. The high-32 bits are loaded into EDX, and the low-32 bits are loaded into EAX. In 64-bit mode, 32-bit register results are zero-extended to 64 bits.

Mnemonic	Opcode	Description
RDPMC	0F 33	Copy the performance monitor counter specified by ECX into EDX:EAX.

The RDPMC instruction always ignores operand size and always uses the 32-bit ECX, EDX, and EAX registers.

The AMD x86-64 Architecture supports four performance counters: PerfCtr0 through PerfCtr3.

Programs running at any privilege level can read performance monitor counters if the PCE flag in CR4 is set.

This instruction is not serializing. Therefore, there is no guarantee that all instructions have completed at the time the performance counter is read.

For more information about performance-counter registers, see Section 13.3.1, “Performance Counters”, on page 416 of Volume 2 of Volume 2 and the documentation for various hardware implementations.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General Protection Fault (GP(0))	X	X	X	The value in ECX is not 0 or 1. CPL is nonzero and CR4.PCE = 0.

RDTSC**Read Time-Stamp Counter**

The RDTSC instruction loads the value of the processor's time-stamp counter into registers EDX:EAX.

Mnemonic	Opcode	Description
RDTSC	0F 31	Copy the time-stamp monitor counter into EDX:EAX.

The time-stamp counter is contained in a 64-bit model-specific register (MSR). The processor sets the counter to 0 upon reset and increments the counter every clock cycle. INIT does not modify the TSC.

The high-order 32 bits are loaded into EDX and the low-order 32 bits are loaded into the EAX register. This instruction ignores operand size always used EDX:EAX.

When the time-stamp disable flag (TSD) in CR4 is set, the RDTSC instruction can only be used at privilege level 0. If the TSD flag is zero, this instruction can be used at any privilege level.

This instruction is not serializing. Therefore, there is no guarantee that all instructions have completed at the time the time-stamp counter is read.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X	The TSD flag in register CR4 is set and the CPL is greater than 0.

RSM**Resume from System Management Mode**

The RSM instruction resumes an operating system or application procedure previously interrupted by a system management interrupt (SMI). The processor state is restored from the information saved when the SMI was taken. If the processor detects invalid state information in the SMM save area, it goes into a shutdown state.

Mnemonic	Opcode	Description
RSM	OF AA	Resume operation of an interrupted program.

Any of the following conditions will cause RSM to shutdown:

- An illegal combination of flags settings in CR0.
- A reserved bit in CR0, CR3, CR4, DR6, DR7 or the extended feature enable register (EFER) is set to 1.
- The following bit setting combination occurs: SMM.EFER.LME = 1, SMM.CR0.PG = 1, SMM.CR4.PAE = 0.

The x86-64 architecture uses a new 64-bit system management mode (SMM) state-save memory image. This 64-bit save-state map is used in all modes, regardless of mode. See Section 10, “System-Management Mode”, on page 331 of Volume 2 for details.

rFLAGS Affected

None. Flags are restored from the state-save map.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The processor is not in the System Management Mode (SMM).

SGDT**Store Global Descriptor Table Register**

Stores the global descriptor table register (GDTR) into the destination operand. In legacy and compatibility mode the destination operand is six bytes; in 64-bit mode it is 10 bytes. In all modes, size prefixes are ignored.

Mnemonic	Opcode	Description
SGDT <i>mem16:32</i>	OF 01 /0	Store global descriptor table register to memory.
SGDT <i>mem16:64</i>	OF 01 /0	Store global descriptor table register to memory.

In legacy or compatibility mode, the lower two bytes of the operand specify the 16-bit limit and the upper 4 bytes specify the lower 32-bits of the base address.

In 64-bit mode, the lower two bytes specify the 16-bit limit and the upper eight bytes specify a 64-bit base address.

Behavior of the SGDT instruction depends on whether the mode is 64-bit mode, but SGDT completely ignores operand size.

This instruction is intended for use in operating system software, but its use in application programs at any CPL is allowed.

Related Instructions

SIDT, SLDT, STR, LGDT, LIDT, LLDT, LTR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The destination operand is a register.
Stack, #SS	X	X	X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X		X X X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered. The destination operand is in a nonwritable segment. There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

SIDT**Store Interrupt Descriptor Table Register**

Stores the interrupt descriptor table register (IDTR) into the destination operand. In legacy and compatibility mode the destination operand is six bytes; in 64-bit mode it is 10 bytes. In all modes, size prefixes are ignored.

Mnemonic	Opcode	Description
SIDT <i>mem16:32</i>	OF 01 /0	Store global descriptor table register to memory.
SIDT <i>mem16:64</i>	OF 01 /0	Store global descriptor table register to memory.

In legacy or compatibility mode, the lower two bytes of the operand specify the 16-bit limit and the upper 4 bytes specify the lower 32-bits of the base address.

In 64-bit mode, the lower two bytes specify the 16-bit limit and the upper eight bytes specify a 64-bit base address.

This instruction is intended for use in operating system software, but its use in application programs is allowed.

Related Instructions

SGDT, SLDT, STR, LGDT, LIDT, LLDT, LTR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The destination operand is a register.
Stack, #SS	X	X	X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X	X	X X X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered. The destination operand is in a nonwritable segment. There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

SLDT

Store Local Descriptor Table Register

The SLDT instruction stores the segment selector from the local descriptor table (LDT) selector to a register or memory destination operand.

Mnemonic	Opcode	Description
SLDT <i>reg16</i>	0F 00 /0	Store local descriptor table register to a 16-bit register.
SLDT <i>reg32</i>	0F 00 /0	Store local descriptor table register to a 32-bit register.
SLDT <i>reg64</i>	0F 00 /0	Store local descriptor table register to a 64-bit register.
SLDT <i>mem16</i>	0F 00 /0	Store local descriptor table register to a 16-bit memory operand.

If the destination operand is a 16-bit memory location, the segment selector is written to memory as a 16-bit value, regardless of operand size.

If the operand is a register, the (16-bit) selector is zero-extended into a 16-, 32-, or 64-bit general purpose register, depending on operand size. If the operand is a 32- or 64-bit register, the segment descriptor is written to the lower 16 bits and the upper bits are filled with zeros.

With a 64-bit operand, this instruction performs the same operation as in legacy mode, except that it zero-extends the 2-byte LDT selector to 64 bits.

The MOV (MOV DS, etc.) instruction can be used more efficiently to load a segment selector to a register or memory.

This instruction can only be issued in protected mode. It can be executed at any privilege level.

Related Instructions

SIDT, SGDT, STR, LIDT, LGDT, LLDT, LTR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

SMSW**Store Machine Status Word**

Stores bits 0 through 15 of the machine status word (CR0). The target can be a 16-bit, 32-bit, or 64-bit register or memory operand or a 16-bit memory operand. With a 32-bit or 64-bit register operand, the low order 16-bits of CR0 are copied into the low order bits of the register with zero extension into the upper 16 or 48 bits.

Mnemonic	Opcode	Description
<i>SMSW reg16</i>	0F 01 /4	Stores the machine status word to a 16-bit register operand.
<i>SMSW reg32</i>	0F 01 /4	Stores the machine status word in the low-order 16 bits of a 32-bit register.
<i>SMSW reg64</i>	0F 01 /4	Stores the machine status word in the low-order 16 bits of a 64-bit register.
<i>SMSW mem16</i>	0F 01 /4	Stores the low order 16 bits of the machine status word in memory.

In 64-bit mode, this instruction performs the same operation as in legacy mode, except that it zero-extends the 2-byte MSW to 64 bits.

It is only provided for compatibility with early processors and should only be used in operating system software. Current practice is to use the MOV(CR*n*) instruction to load all 32 bits of CR0.

This instruction can be used at any privilege level (CPL).

Related Instructions

LMSW, MOV(CR*n*)

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	X	X	X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the data segment limit was exceeded or a non canonical address was encountered.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF		X	X	A page fault resulted from the execution of the instruction.
Alignment check, #AC		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

STI**Set Interrupt Flag**

Sets the interrupt flag (IF) to 1, allowing a maskable hardware interrupt to be recognized through an interrupt request (INTR). INTR is an input signal into the processor, requesting suspension of execution of the current program. It is used for external hardware interrupts. The interrupt flag (IF) is used to mask or unmask the INTR signal. When the INTR is masked (IF=0), the CPU does not acknowledge the interrupt. When the INTR is unmasked (IF=1), the CPU will acknowledge the interrupt request suspend the current process.

Mnemonic	Opcode	Description
STI	FB	Set interrupt flag.

If the current privilege level (CPL) is less than IOPL, STI sets the IF flage to 1. EFLAGS.VIF is set, in virtual mode, if CR4.VME is set (1), or in protected mode, if CR4.PVI is set and CPL = 3. Otherwise, a GP# fault will be generated.

If STI sets IF and it was already clear, then interrupts are not enabled until after the instruction following STI . Thus, if IF is 0, the CLI in the following code sequence has no effect.

```
;; IF=0 before STI!
    STI    ;Set IF=1.
CLI    ;Clear IF.
NOP    ;IF=1! CLI has no effect.
```

In the following sequence, CLI clears IF, since it no longer immediately follows STI.

```
;; IF=0 before STI!
    STI    ;Set IF=1.
NOP    ;IF=1! CLI has no effect.
CLI    ;Clear IF.
```

If STI sets the VIF flag and VIP is already set, a GP# fault will be generated before the next instruction.

Related Instructions

CLI, INT

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
		1/0								1/0						
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. 1/0 is set to either one or zero. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	VIF and VIP are both set.

STR Store Task Register

The STR instruction stores the segment selector from the task register to a 16-, 32-, or 64-bit general-purpose register or 16-bit memory operand.

Mnemonic	Opcode	Description
STR <i>mem16</i>	0F 00 /1	Stores the segment selector from the task register to a 16-bit memory location.
STR <i>reg16</i>	0F 00 /1	Stores the segment selector from the task register to a 16-bit general-purpose register.
STR <i>reg32</i>	0F 00 /1	Stores the segment selector from the task register to a 32-bit general-purpose register.
STR <i>reg64</i>	0F 00 /1	Stores the segment selector from the task register to a 64-bit general-purpose register.

If the destination is a memory location, the segment selector is written to memory as a 16-bit value, regardless of operand size.

If the operand is a 32-bit register, the segment selector is written to the lower 16 bits and the upper bits are filled with zeros.

If the operand is a 64-bit register, this instruction performs the same operation as in legacy mode, except that it zero-extends the 2-byte TR selector to 64 bits.

The STR instruction can only be used in protected mode and it can be used at any privilege level. It is useful only in operation system software.

Related Instructions

LTR, LGDT, LIDT, LLDT, LTR, SIDT, SGDT

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
			X	The destination operand is in a nonwritable segment.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

SWAPGS**Swap GS Register with KernelGSbase MSR**

The SWAPGS instruction provides a fast method for system software to load a pointer to system data-structures. SWAPGS can be used upon entering system-software routines as a result of a SYSCALL instruction, or as a result of an interrupt or exception. Prior to returning to application software, SWAPGS can be used to restore the application's data pointer that was replaced by the system data-structure pointer.

Mnemonic	Opcode	Description
SWAPGS	0F 01 F8	Save GS selector register with KernelGSBase MSR. (Invalid in legacy and compatibility modes.)

The SWAPGS instruction only exchanges the base-address value located in the KernelGSbase model-specific register (MSR address C000_0102h) with the base-address value located in the hidden-portion of the GS selector register (GS.base). This allows the system-kernel software to access kernel data-structures by using the GS segment-override prefix during memory references.

The address stored in the KernelGSbase MSR must be in canonical form. The WRMSR instruction used to load the KernelGSbase MSR causes a general-protection exception if the address loaded is not in canonical form. The SWAPGS instruction itself does not perform a canonical check.

This instruction is only valid in 64-bit mode at CPL=0. A general protection exception is generated if this instruction is executed at any other privilege level.

For additional information about this instruction, refer to Section 6, “System-Management Instructions”, on page 189 of Volume 2.

The opcode for SWAPGS is located in the INVLPG reg space.

Opcode	MOD R/M byte			Instruction	
	MOD	REG	R/M	not 64-bit mode	64-bit mode
0F 01	MOD \Diamond 11	111	xxx	INVLPG	INVLPG
	11	111	000	#UD	SWAPGS
	11	111	\Diamond 000	#UD	#UD

Examples

1. At a kernel entry point, the OS uses SwapGS to obtain a pointer to kernel data structures and simultaneously save the user's GS base. Upon exit, it uses SwapGS to restore the user's GS base:

```

SystemCallEntryPoint:
SwapGS                      ; get kernel pointer, save user GSbase
mov gs:[SavedUserRSP], rsp  ; save user's stack pointer
mov rsp, gs:[KernelStackPtr] ; set up kernel stack
push rax                   ; now save user GPRs on kernel stack
                          . ; perform system service
                          .
SwapGS                      ; restore user GS, save kernel pointer

```

Related Instructions

None.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Software is executing in legacy or compatibility mode.
General protection, #GP			X	CPL > 0.

SYSCALL **Fast System Call**

The SYSCALL and SYSRET instructions are low-latency system call and return control-transfer instructions, which assume that operating system implements a flat-segment memory model. By eliminating unneeded checks, and by loading pre-determined values into the CS and SS segment registers (both visible and hidden portions), calls to and returns from the operating system are greatly simplified. These instructions can be used in any operating mode, but they are particularly well-suited for use in 64-bit mode, which requires implementation of a paged, flat-segment memory model.

Mnemonic	Opcode	Description
SYSCALL	0F 05	Call operating system.

The SYSCALL instruction transfers control to a fixed entry point in an operating system. It is designed for use by system and application software implementing a flat-segment memory model.

This instruction has been optimized by reducing the number of checks and memory references that are normally made so that a call or return takes considerably fewer clock cycles than the CALL FAR /RET FAR instruction method.

It is assumed that the base, limit, and attributes of the Code Segment will remain flat for all processes and for the operating system, and that only the current privilege level for the selector of the calling process should be changed from a current privilege level of three to a new privilege level of zero. It is also assumed (but not checked) that the privilege level of the selectors for the CALL and RET target CS segments are set to zero and three, respectively (for SYSCALL the code segment selector requested privilege level is zero and for SYSRET the code segment selector requested privilege level is three).

The SYSCALL and SYSRET instructions should be used in environments where neither the operating system nor applications alter the Code Segment or Stack Segment in a manner different than that assumed by the instructions. The SYSCALL instruction sets the base, limit, and attributes of both CS and SS to fixed values on entry. The SYSRET instruction sets CS to the same fixed values upon exiting. No memory accesses to descriptors are required. It is the responsibility of the operating system to keep the descriptors in memory that correspond to the CS and SS selectors loaded by the SYSCALL and SYSRET instructions consistent with the segment base, limit, and attribute values forced by these instructions.

Legacy x86 Mode. In legacy x86 mode, when SYSCALL is executed, the EIP register is copied into the ECX register. Bits 31:0 of the 64-bit SYSCALL/SYSRET target address register (STAR) are copied into the EIP register. (The STAR register is model-specific register C000_0081h.)

The IF and VM flags are cleared to 0 to disable interrupts and force the processor out of virtual-8086 mode.

New selectors are loaded, without permission checking (see above), as follows:

- Bits 47:32 of the STAR register specify the selector that is copied into the CS register.
- Bits 47:32 + 1000b of the STAR register specify the selector that is copied into the SS register.
- The CS_base and the SS_base are both cleared to zero.
- The CS_limit and the SS_limit are both set to 4 Gbyte.
- The CS segment attributes are set to execute/read with a CPL of zero.
- The SS segment attributes are set to read/write and expand-up with a 32-bit stack referenced by a 32-bit stack selector.

The CPL is set to 0 regardless of the value of bits 33:32 of the STAR register. There are no permission checks based on the CPL, real mode, or virtual-8086 mode.

The operating system must ensure that the descriptors corresponding to the selectors in the STAR register are consistent with the corresponding on-chip values (such as, base, limit, and attributes) that are set by the SYSCALL and SYSRET instructions.

Long Mode. When long mode is activated, the behavior of the SYSCALL instruction depends on whether the calling software is in 64-bit mode or compatibility mode.

In 64-bit mode, SYSCALL saves the RIP of the instruction following the SYSCALL into RCX and loads the new RIP from LSTAR bits 63:0.

In compatibility mode, SYSCALL saves the RIP of the instruction following the SYSCALL into RCX and loads the new RIP from CSTAR bits 63:0.

- The CS_base and the SS_base are both cleared to zero.
- The CS_limit and the SS_limit are both set to 4 Gbyte.
- The CS segment attributes are set to execute/read for 64-bit code with a CPL of zero.
- The SS segment attributes are set to read/write and expand-up with a 64-bit stack referenced by RSP.

The WRMSR instruction is used to load the target RIP into the LSTAR or CSTAR registers. If the RIP written by WRMSR is not in canonical form, a general-protection exception (#GP) occurs. The SYSCALL and SYSRET CS and SS selectors used when long mode is active are stored in the STAR.

In long mode, the SYSCALL_FLAG_MASK register is used to specify which rFLAGS bits are cleared during a SYSCALL. How SYSCALL and SYSRET handle rFLAGS, depends on the processor's operating mode:

How SYSCALL handles EFLAGS, depends on the processor's operating mode:

In legacy mode, SYSCALL does the following with EFLAGS:

- EFLAGS.IF is cleared to 0.
- EFLAGS.RF is cleared to 0.

In long mode, SYSCALL does the following with EFLAGS:

- The current value of RFLAGS is saved in R11.
- RFLAGS.RF is cleared to 0.
- RFLAGS is masked using the value stored in SYSCALL_FLAG_MASK.

For further details on the SYSCALL and SYSRET instructions and their associated MSR registers (STAR, LSTAR, and CSTAR), see Section 6.1, “Fast System Call and Return”, on page 192 of Volume 2.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

SYSCALL_START:

```
IF (MSR_EFER.SCE = 0)           // check if syscall/sysret are enabled
    EXCEPTION [#UD]
```

```
IF (LONG_MODE)
    SYSCALL_LONG_MODE
ELSE // (LEGACY_MODE)
    SYSCALL_LEGACY_MODE
```

SYSCALL_LONG_MODE:

```
RCX.q = next_RIP
R11.q = RFLAGS    // with rf cleared

IF (64BIT_MODE)
    temp_RIP.q = MSR_LSTAR
ELSE // (COMPATIBILITY_MODE)
    temp_RIP.q = MSR_CSTAR
```



```

CS.sel    = MSR_STAR.SYSCALL_CS AND 0xFFFC
CS.attr   = 64-bit code,dpl0  // always switch to 64-bit mode in long mode
CS.base   = 0x00000000
CS.limit  = 0xFFFFFFFF

```

```

SS.sel    = MSR_STAR.SYSCALL_CS + 8
SS.attr   = 64-bit stack,dpl0
SS.base   = 0x00000000
SS.limit  = 0xFFFFFFFF

```

```

RFLAGS = RFLAGS AND ~MSR_SFmask
RFLAGS.RF = 0

```

```

CPL = 0

```

```

RIP = temp_RIP
EXIT

```

SYSCALL_LEGACY_MODE:

```

RCX.d = next_RIP

```

```

temp_RIP.d = MSR_STAR.eip

```

```

CS.sel    = MSR_STAR.syscall_cs AND 0xFFFC
CS.attr   = 32-bit code,dpl0  // always switch to 32-bit mode in legacy mode
CS.base   = 0x00000000
CS.limit  = 0xFFFFFFFF

```

```

SS.sel    = MSR_STAR.syscall_cs + 8
SS.attr   = 32-bit stack,dpl0
SS.base   = 0x00000000
SS.limit  = 0xFFFFFFFF

```

```

RFLAGS.vm,if,rf=0

```

```

CPL = 0

```

```

RIP = temp_RIP
EXIT

```

Related Instructions

SYSRET, SYSENTER, SYSEXIT

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SYSCALL and SYSRET Instructions bit (bit 11) in CPUID extended function 8000_0001 is cleared to 0.
	X	X	X	The system call extension bit (SCE) of the extended feature enable register (EFER) is set to 0. (The EFER register is MSR C000_0080h.)

SYSENTER System Call

The SYSENTER instruction transfers control to a fixed entry point in an operating system. It is designed for use by system and application software implementing a flat-segment memory model.

Mnemonic	Opcode	Description
SYSENTER	0F 34	Call operating system.

Three model-specific registers (MSRs) are used to specify the target address and stack pointers for the SYSENTER instruction, as well as the CS and SS selectors of the called and returned procedures:

- SYSENTER Target CS: Contains the CS selector of the called procedure. The SS selector is set to `sysenter.cs + 8`.
- SYSENTER Target ESP: Contains the called procedure's stack pointer.
- SYSENTER Target EIP: Contains the offset into the CS of the called procedure.

The hidden portions of the CS and SS segment registers are not loaded from the descriptor table as they would be using a legacy x86 CALL instruction. Instead, the hidden portions are forced by the processor to the following values:

- The CS and SS base values are cleared to 0.
- The CS and SS limit values are set to 4GByte.
- The CS segment attributes are set to execute/read 32-bit code with a CPL of zero.
- The SS segment attributes are set to read/write and expand-up with a 32-bit stack referenced by a 32-bit stack selector.

System software must create corresponding descriptor-table entries referenced by the new CS and SS selectors that match the values described above.

The return EIP and application stack are not saved by this instruction. System software must explicitly save that information.

An invalid-opcode exception occurs if this instruction is used in long mode. Software should use the SYSCALL (and SYSRET) instructions when running in long mode.

For additional information on this instruction, see Section 6.1.3, “SYSENTER and SYSEXIT (Legacy Mode Only)”, on page 195 of Volume 2.

Related Instructions

SYSCALL, SYSEXIT, SYSRET

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
				0						0						
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X X	Software is executing in long mode. The SYSENTER and SYSEXIT Instructions bit (bit 11) in CPUID standard function 1 is cleared to 0.
General protection, #GP	X X			Protected mode is not enabled. The SYSENTER_CS MSR is cleared to 0.

SYSEXIT System Return

The SYSEXIT instruction returns from the operating system to an application. It is a low-latency system return instruction designed for use by system and application software implementing a flat-segment memory model. This is a privileged instruction. The current privilege level must be zero to execute this instruction. An invalid-opcode exception occurs if this instruction is used in long mode. Software should use the SYSRET (and SYSCALL) instructions when running in long mode.

Mnemonic	Opcode	Description
SYSEXIT	0F 35	Return from operating system to application.

When a system procedure performs a SYSEXIT back to application software, the CS selector is updated to point to the second descriptor entry after the SYSENTER CS value (SYSENTER_CS+16). The SS selector is updated to point to the third descriptor entry after the SYSENTER CS value (SYSENTER_CS+24). The CPL is forced to 3, as are the descriptor privilege levels.

The hidden portions of the CS and SS segment registers are not loaded from the descriptor table as they would be using a legacy x86 RET instruction. Instead, the hidden portions are forced by the processor to the following values:

- The CS and SS base values are cleared to 0.
- The CS and SS limit values are set to 4GByte.
- The CS segment attributes are set to 32-bit read/execute, non-conforming at CPL 3.
- The SS segment attributes are set for a 32-bit stack segment, read/write and expand-up.

System software must create corresponding descriptor-table entries referenced by the new CS and SS selectors that match the values described above.

The following additional actions result from executing SYSEXIT:

- EIP is loaded from EDX.
- ESP is loaded from ECX.

System software must explicitly load the return address and application software-stack pointer into the EDX and ECX registers prior to executing SYSEXIT.

For additional information on this instruction, see Section 6.1.3, “SYSENTER and SYSEXIT (Legacy Mode Only)”, on page 195 of Volume 2.

Related Instructions

SYSCALL, SYSENTER, SYSRET

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
					0											
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD			X X	Software is executing in long mode. The SYSENTER and SYSEXIT Instructions bit (bit 11) in CPUID standard function 1 is cleared to 0.
General protection, #GP	X	X	X X	Protected mode is not enabled. CPL>0. The SYSENTER_S MSR is cleared to 0.

SYSRET **Fast System Return**

The SYSCALL and SYSRET instructions are low-latency system call and return control-transfer instructions, which assume that operating system implements a flat-segment memory model. By eliminating unneeded checks, and by loading pre-determined values into the CS and SS segment registers (both visible and hidden portions), calls to and returns from the operating system are greatly simplified. These instructions can be used in any operating mode, but they are particularly well-suited for use in 64-bit mode, which requires implementation of a paged, flat-segment memory model.

Mnemonic	Opcode	Description
SYSRET	0F 07	Return from operating system.

The SYSCALL instruction transfers control to a fixed entry point in an operating system. It is designed for use by system and application software implementing a flat-segment memory model.

This instruction has been optimized by reducing the number of checks and memory references that are normally made so that a call or return takes substantially fewer internal clock cycles when compared to the CALL/RET instruction method.

It is assumed that the base, limit, and attributes of the Code Segment will remain flat for all processes and for the operating system, and that only the current privilege level for the selector of the calling process should be changed from a current privilege level of 0 to a new privilege level of 3. It is also assumed (but not checked) that the privilege level of the selectors for the CALL and RET target CS segments are set to zero and three, respectively (for SYSCALL the code segment selector requested privilege level is zero and for SYSRET the code segment selector requested privilege level is three).

The SYSCALL and SYSRET instructions should be used in environments where neither the operating system nor applications alter the Code Segment or Stack Segment in a manner different than that assumed by the instructions. The SYSCALL instruction sets the base, limit, and attributes of both CS and SS to fixed values on entry. The SYSRET instruction sets CS to the same fixed values upon exiting. In a flat memory model, only the privilege level of the segments need to change. No memory accesses to descriptors are required. It is the responsibility of the operating system to keep the descriptors in memory that correspond to the CS and SS selectors loaded by the SYSCALL and SYSRET instructions consistent with the segment base, limit, and attribute values forced by these instructions.

When a system procedure performs a SYSRET back to application software, the CS selector is updated from bits 63:50 of the STAR register (SYSRET.CS) as follows:

- If the return is to 32-bit mode (legacy or compatibility), CS is updated with the value of SYSRET.CS.
- If the return is to 64-bit mode, CS is updated with the value of SYSRET.CS + 10h.

In both cases, the CPL is forced to 3, effectively ignoring STAR bits 49:48. The SS selector is updated to point to the next descriptor-table entry after the CS descriptor (SYSRET.CS + 8h), and its CPL is also forced to 3.

The hidden portions of the CS and SS segment registers are not loaded from the descriptor table as they would be using a legacy x86 RET instruction. Instead, the hidden portions are forced by the processor to the following values:

- The CS value is cleared to 0.
- The CS limit value is set to 4GByte.
- The CS segment attributes are set to read-only at CPL = 3.
- The SS segment attributes are set to read/write and expand-up.

System software must create corresponding descriptor-table entries referenced by the STAR's SYSRET CS field that match the values described above.

When SYSCALLed system software is running in 64-bit mode, it has been entered from either 64-bit mode or compatibility mode. The corresponding SYSRET needs to know the mode to which it must return. Executing SYSRET in non-64-bit mode or with a 16- or 32-bit operand size, returns to 32-bit mode. Executing SYSRET in 64-bit mode with a 64-bit operand size returns to 64-bit mode.

The instruction pointer is updated with the return address based on the operating mode in which SYSRET is executed:

- If returning to 64-bit mode, SYSRET loads RIP with the value of RCX.
- If returning to 32-bit mode, SYSRET loads EIP with the value of ECX.

How SYSRET handles EFLAGS, depends on the processor's operating mode:

- In long mode, SYSRET loads the lower-32 RFLAGS bits from R11[31:0], and clears the upper 32 RFLAGS bits.
- In legacy mode, SYSRET sets EFLAGS.IF.

For further details on the SYSCALL and SYSRET instructions and their associated MSR registers (STAR, LSTAR, and CSTAR), see Section 6.1, "Fast System Call and Return", on page 192 of Volume 2.

Action

// See Section 2.5.3, “Pseudocode Definitions”, on page 46.

SYSRET_START:

```

    IF (MSR_EFER.SCE = 0)                // check if syscall/sysret are enabled
        EXCEPTION [#UD]

    IF (64BIT_MODE)
        SYSRET_64BIT_MODE
    ELSE // (!64BIT_MODE)
        SYSRET_NON_64BIT_MODE

```

SYSRET_64BIT_MODE:

```

    IF (CPL != 0)                        // sysret requires protected mode, cp10
        EXCEPTION [#GP(0)]

    IF (OPERAND_SIZE = 64)                // return to 64-bit mode
    {
        CS.sel = (MSR_STAR.sysret_cs + 16) OR 3
        CS.base = 0x00000000
        CS.limit = 0xFFFFFFFF
        CS.attr = 64-bit code,dp13

        temp_RIP.q = RCX
    }
    ELSE                                  // return to 32-bit compatibility mode
    {
        CS.sel = MSR_STAR.sysret_cs OR 3
        CS.base = 0x00000000
        CS.limit = 0xFFFFFFFF
        CS.attr = 32-bit code,dp13

        temp_RIP.d = RCX
    }

    SS.sel = MSR_STAR.sysret_cs + 8 // ss selector is changed,
                                   // ss base, limit, attributes unchanged

    RFLAGS.q = R11 // rf=0,vm=0
    CPL = 3

    RIP = temp_RIP
    EXIT

```

SYSRET_NON_64BIT_MODE:

```

    IF ((!PROTECTED_MODE) || (CPL != 0))
        // sysret requires protected mode, cp10

```

```

EXCEPTION [#GP(0)]

CS.sel    = MSR_STAR.sysret_cs OR 3
                                     // return to 32-bit legacy protected mode
CS.base   = 0x00000000
CS.limit  = 0xFFFFFFFF
CS.attr   = 32-bit code,dpl3

temp_RIP.d = RCX

SS.sel    = MSR_STAR.sysret_cs + 8
                                     // ss selector is changed.
                                     // ss base, limit, attributes unchanged

RFLAGS.IF = 1
CPL = 3

RIP = temp_RIP
EXIT

```

Related Instructions

SYSCALL, SYSENTER, SYSEXIT

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Note:
 Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or cleared to zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SYSCALL and SYSRET Instructions bit (bit 11) in CPUID extended function 8000_0001 is cleared to 0.
	X	X	X	The system call extension bit (SCE) of the extended feature enable register (EFER) is set to 0. (The EFER register is MSR C000_0080h.)
General protection, #GP		X	X	CPL is not equal to 0.

UD2 Undefined Operation

Generates an invalid opcode exception. Unlike other undefined opcodes that may be defined as legal instructions in the future, UD2 is guaranteed to stay undefined.

Mnemonic	Opcode	Description
UD2	0F 0B	Raise an invalid opcode exception.

Related Instructions

NOP

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	This instruction is not recognized.

VERR**Verify Segment for Reads**

Verifies whether a code or data segment specified by the segment selector in the 16-bit register or memory operand is readable from the current privilege level. The zero flag (ZF) is set to 1 if the specified segment is readable. Otherwise, ZF is cleared. If the selector references a system descriptor ZF is cleared, since system descriptors are never readable.

Mnemonic	Opcode	Description
VERR <i>reg/mem 16</i>	0F 00 /4	Set the zero flag (ZF) to 1, If the segment selected can be read.

The processor does not recognize the VERR instruction in real or virtual-8086 mode.

Related Instructions

VERW

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
													1/0			
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0

Notes: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or zero is 1/0. Unaffected flags are blank. Undefined flags are U.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

VERW**Verify Segment for Writes**

Verifies whether a code or data segment specified by the segment selector in the 16-bit register or memory operand is writable from the current privilege level. The zero flag (ZF) is set to 1 if the specified segment is writable. Otherwise, ZF is cleared. Since VERW will always clear ZF when checking system descriptors, which are never writable. Code segments can never be verified as writable.

Mnemonic	Opcode	Description
VERW <i>reg/mem16</i>	0F 00 /4	Set the zero flag (ZF) to 1, If the segment selected can be written.

The processor does not recognize the VERR instruction in real or virtual-8086 mode.

Related Instructions**VERR****rFLAGS Affected**

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
													1/0			
21	20	19	18	17	16	14	13–12	11	10	9	8	7	6	4	2	0
Note: Bits 31–22, 15, 5, 3, and 1 are reserved. A flag set to one or zero is 1/0. Unaffected flags are blank. Undefined flags are U.																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		This instruction is not recognized.
Stack, #SS			X	During instruction execution, the stack segment limit was exceeded.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
			X	There is a null segment selector in a data-segment register that is accessing memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment check, #AC			X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

WBINVD**Writeback and Invalidate Caches**

The WBINVD instruction writes all modified cache lines in the internal caches back to main memory and invalidates (flushes) internal caches. It then triggers a bus cycle that causes external caches to invalidate themselves as well. After invalidating internal caches, the processor proceeds immediately with the execution of the next instruction without waiting for external hardware to invalidate its caches.

Mnemonic	Opcode	Description
WBINVD	0F 09	Write modified cache lines to main memory, invalidate internal caches, and trigger external cache flushes.

The INVD instruction can be used when cache coherence with memory is not important.

This instruction never invalidates TLBs.

This instruction is a serializing instruction.

This is a privileged instruction. The current privilege level of a procedure invalidating the processor's internal caches must be zero.

Related Instructions

INVD

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP		X	X	The current privilege level is not equal to 0.

WRMSR

Write to Model-Specific Register

The WRMSR instruction can be used to write data to a 64-bit model-specific register (MSR). These registers are widely used in performance-monitoring and debugging applications, as well as testability and program execution tracing.

Mnemonic	Opcode	Description
WRMSR	0F 30	Write EDX:EAX to the MSR specified by ECX.

This instruction writes the contents of the EDX:EAX register pair into a 64-bit model-specific register specified in the ECX register. The 32 bits in the EDX register are mapped into the high-order bits of the model-specific register and the 32 bits in EAX form the low-order 32 bits.

This instruction must be executed at a privilege level of 0 or a general protection fault #GP(0) will be raised. This exception is also generated if an attempt is made to specify a reserved or unimplemented model-specific register in ECX.

This is a serializing instruction.

The CPUID instruction can provide model information usefull in determining the existence of a particular MSR.

See Volume 2, *System Programming*, for more information about model specific registers, machine check architecture, performance monitoring and debug registers.

For more information about model-specific registers, see Volume 2, *Ssystem Programming*, Appendix A “Machine Check Cross Reference” and the documentation for various hardware implementations.

Related Instructions

RDMSR.

rFLAGS Affected

None.

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	X		X	The value in ECX specifies a reserved or unimplemented MSR address.
		X	X	The current privilege level is not 0.
	X		X	Writing 1 to any bit that must be zero (MBZ) in the MSR.

3 Opcode and Operand Encodings

This section specifies the hexadecimal and/or binary encodings for the opcodes and the implicit operand references used in the x86-64 instruction set. For an overview of the instruction formats to which these encodings apply, see Section 1, “Instruction Formats”, on page 1.

3.1 Opcode-Syntax Notation

The following notation is used in this section to specify opcodes and their operands:

- A* Direct address of operand is encoded in instruction without a ModRM byte. Complex addressing using the SIB byte cannot be done.
- C* Control register specified by the ModRM *reg* field.
- D* Debug register specified by the ModRM *reg* field.
- E* The opcode is followed by a ModRM byte that specifies a GPR or memory operand. Memory addresses can be computed from a segment register, SIB byte, and/or displacement.
- F* rFLAGS register.
- G* The ModRM *reg* field selects a GPR.
- I* Immediate value.
- J* The instruction includes a relative offset that is added to the rIP.
- M* A memory reference encoded in the ModRM byte.
- O* The offset of an operand is encoded in the instruction. There is no ModRM byte in the instruction. Complex addressing using the SIB byte cannot be done.
- P* The ModRM *reg* field specifies a 64-bit MMX register.
- Q* A ModRM byte specifies an MMX-register or memory operand. Memory addresses can be computed from a segment register, SIB byte, and/or displacement.

- R* The ModRM *mod* field must be 11b (i.e., only the register form of this field is valid).
- S* The ModRM *reg* field specifies a segment register.
- T* The ModRM *reg* field specifies a test register.
- V* The ModRM *reg* field specifies a 128-bit XMM register.
- W* A ModRM byte specifies an XMM or memory operand. Memory addresses can be computed from a segment register, SIB byte, and/or displacement.
- X* A memory operand is addressed by the DS.rSI registers. Used in string instructions.
- Y* A memory operand is addressed by the ES.rDI registers. Used in string instructions.
- a* Two 16-bit or 32-bit memory operands, depending on the effective operand size (used in BOUND).
- b* A byte, irrespective of the effective operand size.
- c* A byte or word, depending on the effective operand size.
- d* A doubleword (32 bits), irrespective of the effective operand size.
- dq* A double-quadword (128 bits), irrespective of the effective operand size.
- p* A 32-bit or 48-bit pointer, depending on the effective operand size.
- pd* A 128-bit double-precision floating-point vector operand (packed double).
- pi* A 64-bit MMX operand (packed integer).
- ps* A 128-bit single-precision floating-point vector operand (packed single).
- q* A quadword, irrespective of the effective operand size.
- s* A 6-byte pseudo-descriptor.
- sd* A scalar double-precision floating-point operand (scalar double).

- si* A scalar doubleword (32-bit) integer operand (scalar integer).
- ss* A scalar single-precision floating-point operand (scalar single).
- v* A word, doubleword, or quadword, depending on the effective operand size.
- w* A word, irrespective of the effective operand size.
- z* A word if the effective operand size is 16 bits, or a doubleword if the effective operand size is 32 or 64 bits.
- /n* A ModRM-byte *reg* field or SIB-byte *base* field that contains a value (*n*) between zero (binary 000) and 7 (binary 111).

For definitions of the mnemonics used to name registers, see Section 1.3, “Summary of Registers and Data Types”, on page 28.

3.2 Opcode Encodings

3.2.1 One-Byte Opcodes

Table 3-1 shows the one-byte opcodes in which the low nibble is in the range 0:7h. Table 3-2 on page 1127 shows those opcodes in which the low nibble is in the range 8:Fh. In both tables, the rows show the full range (0:Fh) of the high nibble, and the columns show the specified range of the low nibble.

Table 3-1. One-Byte Opcodes, Low Nibble 0:7h

Nibble ¹	0	1	2	3	4	5	6	7
0	ADD Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						PUSH ES ³	POP ES ³
1	ADC Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						PUSH SS ³	POP SS ³
2	AND Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAx, Iz						seg ES ³	DAA ³
3	XOR Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						seg SS ³	AAA ³
4	INC ⁵ rAX rCX rDX rBX rSP rBP rSI rDI							
5	PUSH rAX/r8 rCX/r9 rDX/r10 rBX/r11 rSP/r12 rBP/r13 rSI/r14 rDI/r15							
6	PUSHA/D ³	POPA/D ³	BOUND ³ Gv, Ma	ARPL ³ Ew, Rw MOVSD ⁴	seg FS	seg GS	operand size	address size
7	JO Jb	JNO Jb	JB Jb	JNB Jb	JZ Jb	JNZ Jb	JBE Jb	JNBE Jb
8	Group 1 ² Eb, Ib Ev, Iz Eb, Ib ³ Ev, Ib				TEST Eb, Gb Ev, Gv		XCHG Eb, Gb Ev, Gv	
9	NOP rAX/r8, rAX	XCHG rCX/r9, rAX rDX/r10, rAX rBX/r11, rAX rSP/r12, rAX rBP/r13, rAX rSI/r14, rAX rDI/r15, rAX						
A	MOV AL, Ob rAX, Ov Ob, AL Ov, rAX				MOVSB Xb, Yb	MOVSW/D/Q Xv, Yv	CMPSB Xb, Yb	CMPSW/D/Q Xv, Yv
B	MOV AL, Ib CL, Ib DL, Ib BL, Ib AH, Ib CH, Ib DH, Ib BH, Ib							

Note:

1. Rows in this table show the high opcode nibble, columns show the low opcode nibble.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.
4. Valid only in 64-bit mode.
5. Used as REX prefixes in 64-bit mode.

Table 3-1. One-Byte Opcodes, Low Nibble 0:7h (continued)

Nibble ¹	0	1	2	3	4	5	6	7
C	Group 2 ²		RET near		LES ³	LDS ³	Group 11 ²	
	Eb, lb	Ev, lb	lw		Gv, Mp	Gv, Mp	Eb, lb	Ev, lz
D	Group 2 ²				AAM ³	AAD ³	SALC ³	XLAT
	Eb, 1	Ev, 1	Eb, CL	Ev, CL				
E	LOOPNE/NZ	LOOPE/Z	LOOP	JrCXZ	IN		OUT	
	Jb	Jb	Jb	Jb	AL, lb	eAX, lb	lb, AL	lb, eAX
F	LOCK:	INT1	REPNE:	REP:	HLT	CMC	Group 3 ²	Group 3 ²
		ICE Bkpt		REPE:			Eb	Ev

Note:

1. Rows in this table show the high opcode nibble, columns show the low opcode nibble.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.
4. Valid only in 64-bit mode.
5. Used as REX prefixes in 64-bit mode.

Table 3-2. One-Byte Opcodes, Low Nibble 8:Fh

Nibble ¹	8	9	A	B	C	D	E	F
0	OR Eb, Gb						PUSH CS ³	2-byte opcodes
1	SBB Eb, Gb						PUSH DS ³	POP DS ³
2	SUB Eb, Gb						seg CS ³	DAS ³
3	CMP Eb, Gb						seg DS ³	AAS ³
4	DEC ⁵ rAX							
5	POP rAX/r8							
6	PUSH lz	IMUL Gv, Ev, lz	PUSH lb	IMUL Gv, Ev, lb	INSB Yb, DX	INSD Yv, DX	OUTSB DX, Xb	OUTSD DX, Xv
7	JS Jb	JNS Jb	JP Jb	JNP Jb	JL Jb	JNL Jb	JLE Jb	JNLE Jb

Note:

1. Rows in this table show the high opcode nibble, columns show the low opcode nibble.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.
4. Valid only in 64-bit mode.
5. Used as REX prefixes in 64-bit mode.

Table 3-2. One-Byte Opcodes, Low Nibble 8:Fh (continued)

Nibble ¹	8	9	A	B	C	D	E	F
8	Eb, Gb	Ev, Gv	MOV Gb, Eb	Gv, Ev	Ew, Sw	LEA Gv, M	MOV Sw, Ew	Group 1a ²
9	CBW, CWDE CDQE	CWD, CDQ, CQO	CALL ³ Ap	WAIT FWAIT	PUSHF/D/Q Fv	POPF/D/Q Fv	SAHF ³	LAHF ³
A	TEST AL, Ib	rAX, Iz	STOSB Yb, AL	STOSW/D/Q Yv, rAX	LODSB AL, Xb	LODSW/D/Q rAX, Xv	SCASB AL, Yb	SCASW/D/Q rAX, Yv
B	MOV							
	rAX, Iv	rCX, Iv	rDX, Iv	rBX, Iv	rSP, Iv	rBP, Iv	rSI, Iv	rDI, Iv
C	ENTER Iw, Ib	LEAVE	RET far Iw	RET far	INT3	INT Ib	INTO ³	IRET, IRETD IRETQ
D	x87 see Table 3-10 on page 1145							
E	CALL Jz	Jz	JMP Ap ³	Jb	IN AL, DX	eAX, DX	OUT DX, AL	DX, eAX
F	CLC	STC	CLI	STI	CLD	STD	Group 4 ² INC, DEC	Group 5 ² INC, DEC etc.

Note:

1. Rows in this table show the high opcode nibble, columns show the low opcode nibble.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.
4. Valid only in 64-bit mode.
5. Used as REX prefixes in 64-bit mode.

3.2.2 Two-Byte Opcodes

All two-byte opcodes have 0Fh as their first byte. Table 3-3 on page 1129 shows the second byte of the two-byte opcodes in which the second-byte's low nibble is in the range 0:7h. Table 3-4 on page 1133 shows those opcodes in which the second-byte's low nibble is in the range 8:Fh. In both tables, the rows show the full range (0:Fh) of the high nibble, and the columns show the low nibble of the opcode. The left-most column shows special-purpose prefix bytes used in many 128-bit and 64-bit instructions to modify the opcode.

Table 3-3. Second Byte of Two-Byte Opcodes, Low Nibble 0:7h

Prefix	Nibble ¹	0	1	2	3	4	5	6	7
none	0	Group 6 ²	Group 7 ²	LAR Gv, Ew	LSL Gv, Ew	invalid	SYSCALL	CLTS	SYSRET
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	1	MOVUPS		MOVLPS, MOVHLPs	MOVLPS	UNPCKLPS	UNPCKHPS	MOVHPS, MOVLHPS	MOVHPS
		Vps, Wps	Wps, Vps	Vps, Wps	Wps, Vps	Vps, Wq	Vps, Wq	Vps, Wps	Wps, Vps
F3		MOVSS Vss, Wss	MOVSS Wss, Vss	invalid	invalid	invalid	invalid	invalid	invalid
66		MOVUPD Vpd, Wpd	MOVUPD Wpd, Vpd	MOVLPD Vq, Mq	MOVLPD Mq, Vq	UNPCKLPD Vpd, Wq	UNPCKHPD Vpd, Wq	MOVHPD Vq, Mq	MOVHPD Mq, Vq
F2		MOVSD Vsd, Wsd	MOVSD Wsd, Vsd	invalid	invalid	invalid	invalid	invalid	invalid
none	2	MOV				invalid	invalid	invalid	invalid
		Rd/q, Cd/q	Rd/q, Dd/q	Cd/q, Rd/q	Dd/q, Rd/q				
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	3	WRMSR	RDTSC	RDMSR	RDPMC	SYSENTER ³	SYSEXIT ³	invalid	invalid
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows in the table show the high nibble of the second opcode bytes, columns show the low nibble of this byte.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.

Table 3-3. Second Byte of Two-Byte Opcodes, Low Nibble 0:7h (continued)

Prefix	Nibble ¹	0	1	2	3	4	5	6	7
none	4	CMOVO Gv, Ev	CMOVNO Gv, Ev	CMOVNB/C /NAE Gv, Ev	CMOVAE /NB/NC Gv, Ev	CMOVE/Z Gv, Ev	CMOVNE /NZ Gv, Ev	CMOVBE /NA Gv, Ev	CMOVA /NBE Gv, Ev
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	5	MOVMSKPS Ed, Vps	SQRTPS Vps, Wps	RSQRTPS Vps, Wps	RCPPS Vps, Wps	ANDPS Vps, Wps	ANDNPS Vps, Wps	ORPS Vps, Wps	XORPS Vps, Wps
F3		invalid	SQRTSS Vss, Wss	RSQRTSS Vss, Wss	RCPSS Vss, Wss	invalid	invalid	invalid	invalid
66		MOVMSKPD Ed, Vpd	SQRTPD Vpd, Wpd	invalid	invalid	ANDPD Vpd, Wpd	ANDNPD Vpd, Wpd	ORPD Vpd, Wpd	XORPD Vpd, Wpd
F2		invalid	SQRTSD Vsd, Wsd	invalid	invalid	invalid	invalid	invalid	invalid
none	6	PUNPCKLBW Pq, Qd	PUNPCKLWD Pq, Qd	PUNPCKLDQ Pq, Qd	PACKSSWB Pq, Qq	PCMPGTB Pq, Qq	PCMPGTW Pq, Qq	PCMPGTD Pq, Qq	PACKUSWB Pq, Qq
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		PUNPCKLBW Vdq, Wq	PUNPCKLWD Vdq, Wq	PUNPCKLDQ Vdq, Wq	PACKSSWB Vdq, Wdq	PCMPGTB Vdq, Wdq	PCMPGTW Vdq, Wdq	PCMPGTD Vdq, Wdq	PACKUSWB Vdq, Wdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	7	PSHUFW Pq, Qq, Ib	Group 12 ²	Group 13 ²	Group 14 ²	PCMPEQB Pq, Qq	PCMPEQW Pq, Qq	PCMPEQD Pq, Qq	EMMS
F3		PSHUFHW Vq, Wq, Ib	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		PSHUFD Vdq, Wdq, Ib	Group 12 ²	Group 13 ²	Group 14 ²	PCMPEQB Vdq, Wdq	PCMPEQW Vdq, Wdq	PCMPEQD Vdq, Wdq	invalid
F2		PSHUFLW Vq, Wq, Ib	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an *OFh* byte. Rows in the table show the high nibble of the second opcode bytes, columns show the low nibble of this byte.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.

Table 3-3. Second Byte of Two-Byte Opcodes, Low Nibble 0:7h (continued)

Prefix	Nibble ¹	0	1	2	3	4	5	6	7
none	8	JO Jz	JNO Jz	JB Jz	JNB Jz	JZ Jz	JNZ Jz	JBE Jz	JNBE Jz
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	9	SETO Eb	SETNO Eb	SETB Eb	SETNB Eb	SETZ Eb	SETNZ Eb	SETBE Eb	SETNBE Eb
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	A	PUSH FS	POP FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL	invalid	invalid
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	B	CMPXCHG Eb, Gb		LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb	MOVZX Gv, Ew
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows in the table show the high nibble of the second opcode bytes, columns show the low nibble of this byte.
2. An opcode extension is specified in bits 5:3 of the ModRM byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.

Table 3-3. Second Byte of Two-Byte Opcodes, Low Nibble 0:7h (continued)

Prefix	Nibble ¹	0	1	2	3	4	5	6	7
none	C	XADD Eb, Gb		CMPPS Vps, Wps, Ib	MOVNTI Md, Gd	PINSRW Pq, Ed, Ib	PEXTRW Gd, Pq, Ib	SHUFPS Vps, Wps, Ib	Group 9 ²
F3		invalid	invalid	CMPSS Vss, Wss, Ib	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	CMPPD Vpd, Wpd, Ib	invalid	PINSRW Vdq, Ed, Ib	PEXTRW Gd, Vdq, Ib	SHUPD Vpd, Wpd, Ib	invalid
F2		invalid	invalid	CMPSD Vsd, Wsd, Ib	invalid	invalid	invalid	invalid	invalid
none	D	invalid	PSRLW Pq, Qq	PSRLD Pq, Qq	PSRLQ Pq, Qq	PADDQ Pq, Qq	PMULLW Pq, Qq	invalid	PMOVMASKB Gd, Pq
F3		invalid	invalid	invalid	invalid	invalid	invalid	MOVQ2DQ Vq, Pq	invalid
66		invalid	PSRLW Vdq, Wdq	PSRLD Vdq, Wdq	PSRLQ Vdq, Wdq	PADDQ Vdq, Wdq	PMULLW Vdq, Wdq	MOVQ Wq, Vq	PMOVMASKB Gd, Vq
F2		invalid	invalid	invalid	invalid	invalid	invalid	MOVDQ2Q Pq, Vq	invalid
none	E	PAVGB Pq, Qq	PSRAW Pq, Qq	PSRAD Pq, Qq	PAVGW Pq, Qq	PMULHUW Pq, Qq	PMULHW Pq, Qq	invalid	MOVNTQ Mq, Pq
F3		invalid	invalid	invalid	invalid	invalid	invalid	CVTDQ2PD Vpd, Wq	invalid
66		PAVGB Vdq, Wdq	PSRAW Vdq, Wdq	PSRAD Vdq, Wdq	PAVGW Vdq, Wdq	PMULHUW Vdq, Wdq	PMULHW Vdq, Wdq	CVTTPD2DQ Vq, Wpd	MOVNTDQ Mdq, Vdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	CVTPD2DQ Vq, Wpd	invalid
none	F	invalid	PSLLW Pq, Qq	PSLLD Pq, Qq	PSLLQ Pq, Qq	PMULUDQ Pq, Qq	PMADDWD Pq, Qq	PSADBQ Pq, Qq	MASKMOVQ Pq, Pq
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	PSLLW Vdq, Wdq	PSLLD Vdq, Wdq	PSLLQ Vdq, Wdq	PMULUDQ Vdq, Wdq	PMADDWD Vdq, Wdq	PSADBQ Vdq, Wdq	MASK- MOVQDU Vdq, Vdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an *OFh* byte. Rows in the table show the high nibble of the second opcode bytes, columns show the low nibble of this byte.
2. An opcode extension is specified in bits 5:3 of the *ModRM* byte. See Section 3.2.4 on page 1138 for details.
3. Invalid in 64-bit mode.

Table 3-4. Second Byte of Two-Byte Opcodes, Low Nibble 8:Fh

Prefix	Nibble ¹	8	9	A	B	C	D	E	F
none	0	INVD	WBINVD	invalid	UD2	invalid	PREFETCH	FEMMS	3DNow! See Section 3.2.6 on page 1141
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	1	Group P ²	NOP	NOP	NOP	NOP	NOP	NOP	NOP
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	2	MOVAPS Vps, Wps Wps, Vps		CVTPI2PS Vps, Qq	MOVNTPS Wps, Vps	CVTTPS2PI Pq, Wps	CVTPS2PI Pq, Wps	UCOMISS Vss, Wss	COMISS Vps, Wps
F3		invalid	invalid	CVTSI2SS Vss, Ed	invalid	CVTTSS2SI Gd, Wss	CVTSS2SI Gd, Wss	invalid	invalid
66		MOVAPD Vpd, Wpd	MOVAPD Wpd, Vpd	CVTPI2PD Vpd, Qq	MOVNTPD Mpd, Vpd	CVTTPD2PI Qq, Wpd	CVTPD2PI Qq, Wpd	UCOMISD Vsd, Wsd	COMISD Vpd, Wsd
F2		invalid	invalid	CVTSI2SD Vsd, Ed	invalid	CVTTSD2SI Gd, Wsd	CVTSD2SI Gd, Wsd	invalid	invalid
none	3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows show high opcode nibble (hex), columns show low opcode nibble, in hex.
2. An opcode extension is specified in the ModRM reg field (bits 5:3). See Section 3.2.4 on page 1138 for details.

Table 3-4. Second Byte of Two-Byte Opcodes, Low Nibble 8:Fh (continued)

Prefix	Nibble ¹	8	9	A	B	C	D	E	F
none	4	CMOVS Gv, Ev	CMOVNS Gv, Ev	CMOVP/PE Gv, Ev	CMOVNP/PO Gv, Ev	CMOVL/NGE Gv, Ev	CMOVGE/NL Gv, Ev	CMOVLE/NG Gv, Ev	CMOVG/NLE Gv, Ev
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	5	ADDPS Vps, Wps	MULPS Vps, Wps	CVTSP2PD Vpd, Wps	CVTDQ2PS Vps, Wdq	SUBPS Vps, Wps	MINPS Vps, Wps	DIVPS Vps, Wps	MAXPS Vps, Wps
F3		ADDSS Vss, Wss	MULSS Vss, Wss	CVTSS2SD Vsd, Wss	CVTTPS2DQ Vdq, Wps	SUBSS Vss, Wss	MINSS Vss, Wss	DIVSS Vss, Wss	MAXSS Vss, Wss
66		ADDPD Vpd, Wpd	MULPD Vpd, Wpd	CVTPD2PS Vps, Wpd	CVTPS2DQ Vdq, Wps	SUBPD Vpd, Wpd	MINPD Vpd, Wpd	DIVPD Vpd, Wpd	MAXPD Vpd, Wpd
F2		ADDSD Vsd, Wsd	MULSD Vsd, Wsd	CVTSD2SS Vss, Wsd	invalid	SUBSD Vsd, Wsd	MINSD Vsd, Wsd	DIVSD Vsd, Wsd	MAXSD Vsd, Wsd
none	6	PUNPCK- HBW Pq, Qd	PUNPCK- HWD Pq, Qd	PUNPCK- HDQ Pq, Qd	PACKSSDW Pq, Qq	invalid	invalid	MOVD Pq, Ed	MOVQ Pq, Qq
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	MOVDQU Vdq, Wdq
66		PUNPCK- HBW Vdq, Wq	PUNPCK- HWD Vdq, Wq	PUNPCK- HDQ Vdq, Wq	PACKSSDW Vdq, Wdq	PUNPCK- LQDQ Vdq, Wq	PUNPCK- HQDQ Vdq, Wq	MOVD Vq, Ed	MOVDQA Vdq, Wdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	7	invalid	invalid	invalid	invalid	invalid	invalid	MOVD Ed, Pd	MOVQ Qq, Pq
F3		invalid	invalid	invalid	invalid	invalid	invalid	MOVQ Vq, Wq	MOVDQU Wdq, Vdq
66		invalid	invalid	invalid	invalid	invalid	invalid	MOVD Ed, Vd	MOVDQA Wdq, Vdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows show high opcode nibble (hex), columns show low opcode nibble, in hex.
2. An opcode extension is specified in the ModRM reg field (bits 5:3). See Section 3.2.4 on page 1138 for details.

Table 3-4. Second Byte of Two-Byte Opcodes, Low Nibble 8:Fh (continued)

Prefix	Nibble ¹	8	9	A	B	C	D	E	F
none	8	JS Jz	JNS Jz	JP Jz	JNP Jz	JL Jz	JNL Jz	JLE Jz	JNLE Jz
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	9	SETS Eb	SETNS Eb	SETP Eb	SETNP Eb	SETL Eb	SETNL Eb	SETLE Eb	SETNLE Eb
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	A	PUSH GS	POP GS	RSM	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	Group 15 ²	IMUL Gv, Ev
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	B	invalid	Group 10 ² UD2	Group 8 ² Ev, Ib	BTC Ev, Gv	BSF Gv, Ev	BSR Gv, Ev	MOVSX Gv, Eb	MOVSX Gv, Ew
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows show high opcode nibble (hex), columns show low opcode nibble, in hex.
2. An opcode extension is specified in the ModRM reg field (bits 5:3). See Section 3.2.4 on page 1138 for details.

Table 3-4. Second Byte of Two-Byte Opcodes, Low Nibble 8:Fh (continued)

Prefix	Nibble ¹	8	9	A	B	C	D	E	F
none	C	BSWAP EAX/r8	BSWAP ECX/r9	BSWAP EDX/r10	BSWAP EBX/r11	BSWAP ESP/r12	BSWAP EBP/r13	BSWAP ESI/r14	BSWAP EDI/r15
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	D	PSUBUSB Pq, Qq	PSUBUSW Pq, Qq	PMINUB Pq, Qq	PAND Pq, Qq	PADDUSB Pq, Qq	PADDUSW Pq, Qq	PMAXUB Pq, Qq	PANDN Pq, Qq
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		PSUBUSB Vdq, Wdq	PSUBUSW Vdq, Wdq	PMINUB Vdq, Wdq	PAND Vdq, Wdq	PADDUSB Vdq, Wdq	PADDUSW Vdq, Wdq	PMAXUB Vdq, Wdq	PANDN Vdq, Wdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	E	PSUBSB Pq, Qq	PSUBSW Pq, Qq	PMINSW Pq, Qq	POR Pq, Qq	PADDSB Pq, Qq	PADDSW Pq, Qq	PMAXSW Pq, Qq	PXOR Pq, Qq
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		PSUBSB Vdq, Wdq	PSUBSW Vdq, Wdq	PMINSW Vdq, Wdq	POR Vdq, Wdq	PADDSB Vdq, Wdq	PADDSW Vdq, Wdq	PMAXSW Vdq, Wdq	PXOR Vdq, Wdq
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
none	F	PSUBB Pq, Qq	PSUBW Pq, Qq	PSUBD Pq, Qq	PSUBQ Pq, Qq	PADDB Pq, Qq	PADDW Pq, Qq	PADDD Pq, Qq	invalid
F3		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
66		PSUBB Vdq, Wdq	PSUBW Vdq, Wdq	PSUBD Vdq, Wdq	PSUBQ Vdq, Wdq	PADDB Vdq, Wdq	PADDW Vdq, Wdq	PADDD Vdq, Wdq	invalid
F2		invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. All two-byte opcodes begin with an 0Fh byte. Rows show high opcode nibble (hex), columns show low opcode nibble, in hex.
2. An opcode extension is specified in the ModRM reg field (bits 5:3). See Section 3.2.4 on page 1138 for details.

3.2.3 rFLAGS Condition Codes for Two-Byte Opcodes

Table 3-5 shows the rFLAGS condition codes specified by the low nibble in the second opcode byte of the CMOVcc, Jcc, and SETcc instructions.

Table 3-5. rFLAGS Condition Codes for CMOVcc, Jcc, and SETcc

Low Nibble of Second Opcode Byte (hex)	rFLAGS Value	cc Mnemonic	Arithmetic Type	Condition(s)
0	OF = 1	O		Overflow
1	OF = 0	NO		No Overflow
2	CF = 1	B, C, NAE	Unsigned	Below, Carry, Not Above or Equal
3	CF = 0	NB, NC, AE		Not Below, No Carry, Above or Equal
4	ZF = 1	E, Z		Equal, Zero
5	ZF = 0	NE, NZ		Not Equal, Not Zero
6	CF = 1 or ZF = 1	BE, NA		Below or Equal, Not Above
7	CF = 0 or ZF = 0	NBE, A		Not Below or Equal, Above
8	SF = 1	S		Sign
9	SF = 0	NS		Not Sign
A	PF = 1	P, PE, U	Signed	Parity, Parity Even, Unordered
B	PF = 0	NP, PO, NU		Not Parity, Parity Odd, Not Unordered
C	(SF xor OF) = 1	L, NGE		Less than, Not Greater than or Equal to
D	(SF xor OF) = 0	NL, GE		Not Less than, Greater than or Equal to
E	(SF xor OF) = 1 or ZF = 1	LE, NG		Less than or Equal to, Not Greater than
F	(SF xor OF) = 0 or ZF = 0	NLE, G		Not Less than or Equal to, Greater than

3.2.4 ModRM Extensions to One-Byte and Two-Byte Opcodes

The ModRM byte, which immediately follows the last opcode byte, is used in certain instruction encodings to provide additional opcode bits with which to define the instruction's function. ModRM bytes have three fields—*mod*, *reg*, and *r/m*, as shown in Figure 3-2.

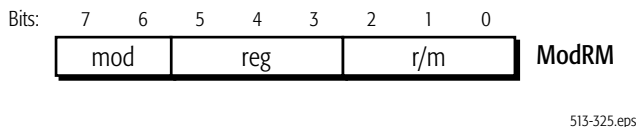


Figure 3-1. ModRM-Byte *reg* Field

In most cases, the *reg* field (bits 5:3) provides the additional bits with which to extend the encodings of the first one or two opcode bytes. In the case of the 64-bit media instructions and many of the x87 floating-point instructions, the entire ModRM byte is used to extend the opcode encodings.

Table 3-6 shows how the ModRM *reg* field is used to extend the range of one-byte and two-byte opcodes. In all cases, the binary value in the ModRM *mod* field (bits 7:6) is 11. The opcode ranges are organized into *groups* of opcode extensions. The group number is shown in the left-most column of Table 3-6. These groups are referenced in the opcodes shown in Table 3-1 on page 1126 through Table 3-4 on page 1133. An entry of “n.a.” in the Prefix column means that prefixes are not applicable to the opcodes in that row. Prefixes only apply to certain 128-bit media, 64-bit media, and a few other instructions introduced with the SSE or SSE-2 technologies.

The /0 through /7 notation for the ModRM *reg* field (bits 5:3) means that the three-bit field contains a value from zero (binary 000) to 7 (binary 111).

Table 3-6. One-Byte and Two-Byte Opcode ModRM Extensions

Group Number	Opcode	Prefix	ModRM <i>reg</i> Field							
			/0	/1	/2	/3	/4	/5	/6	/7
Group 1	80	n.a.	ADD Eb, Ib	OR Eb, Ib	ADC Eb, Ib	SBB Eb, Ib	AND Eb, Ib	SUB Eb, Ib	XOR Eb, Ib	CMP Eb, Ib
	81		ADD Ev, Iz	OR Ev, Iz	ADC Ev, Iz	SBB Ev, Iz	AND Ev, Iz	SUB Ev, Iz	XOR Ev, Iz	CMP Ev, Iz
	82		ADD Eb, Ib ²	OR Eb, Ib ²	ADC Eb, Ib ²	SBB Eb, Ib ²	AND Eb, Ib ²	SUB Eb, Ib ²	XOR Eb, Ib ²	CMP Eb, Ib ²
	83		ADD Ev, Ib	OR Ev, Ib	ADC Ev, Ib	SBB Ev, Ib	AND Ev, Ib	SUB Ev, Ib	XOR Ev, Ib	CMP Ev, Ib
Group 1a	8F	n.a.	POP Ev	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 2	C0	n.a.	ROL Eb, Ib	ROR Eb, Ib	RCL Eb, Ib	RCR Eb, Ib	SHL/SAL Eb, Ib	SHR Eb, Ib	SHL/SAL Eb, Ib	SAR Eb, Ib
	C1		ROL Ev, Ib	ROR Ev, Ib	RCL Ev, Ib	RCR Ev, Ib	SHL/SAL Ev, Ib	SHR Ev, Ib	SHL/SAL Ev, Ib	SAR Ev, Ib
	D0		ROL Eb, 1	ROR Eb, 1	RCL Eb, 1	RCR Eb, 1	SHL/SAL Eb, 1	SHR Eb, 1	SHL/SAL Eb, 1	SAR Eb, 1
	D1		ROL Ev, 1	ROR Ev, 1	RCL Ev, 1	RCR Ev, 1	SHL/SAL Ev, 1	SHR Ev, 1	SHL/SAL Ev, 1	SAR Ev, 1
	D2		ROL Eb, CL	ROR Eb, CL	RCL Eb, CL	RCR Eb, CL	SHL/SAL Eb, CL	SHR Eb, CL	SHL/SAL Eb, CL	SAR Eb, CL
	D3		ROL Ev, CL	ROR Ev, CL	RCL Ev, CL	RCR Ev, CL	SHL/SAL Ev, CL	SHR Ev, CL	SHL/SAL Ev, CL	SAR Ev, CL
Group 3	F6	n.a.	TEST Ib		NOT Eb	NEG Eb	MUL Eb	IMUL AL	DIV AL	IDIV AL
	F7		TEST Iz		NOT Ev	NEG Ev	MUL rAX	IMUL rAX	DIV rAX	IDIV rAX
Group 4	FE	n.a.	INC Eb	DEC Eb	invalid	invalid	invalid	invalid	invalid	invalid
Group 5	FF	n.a.	INC Ev	DEC Ev	CALL Ev	CALL Ep	JMP Ev	JMP Ep	PUSH Ev	invalid
Group 6	0F 00	n.a.	SLDT Ew	STR Ew	LLDT Ew	LTR Ew	VERR Ew	VERW Ew	invalid	invalid

Note:

1. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SWAPGS.
2. Invalid in 64-bit mode.
3. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SFENCE.
4. This instruction takes a ModRM byte.
5. Reserved prefetch encodings are aliased to the /0 encoding (PREFETCH Exclusive) if otherwise undefined.

Table 3-6. One-Byte and Two-Byte Opcode ModRM Extensions (continued)

Group Number	Opcode	Prefix	ModRM <i>reg</i> Field							
			/0	/1	/2	/3	/4	/5	/6	/7
Group 7	0F 01	n.a.	SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Ew	invalid	LMSW Ew	INVLPG Mb SWAPGS ¹
Group 8	0F BA	n.a.	invalid	invalid	invalid	invalid	BT	BTS	BTR	BTC
Group 9	0F C7	n.a.	invalid	CMPXCHG Mq	invalid	invalid	invalid	invalid	invalid	invalid
Group 10	0F B9	n.a.	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 11	C6:C7	n.a.	MOV	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 12	0F 71	none	invalid	invalid	PSRLW Pq, Ib	invalid	PSRAW Pq, Ib	invalid	PSLLW Pq, Ib	invalid
		66	invalid	invalid	PSRLW Vdq, Ib	invalid	PSRAW Vdq, Ib	invalid	PSLLW Vdq, Ib	invalid
		F2, F3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 13	0F 72	none	invalid	invalid	PSRLD Pq, Ib	invalid	PSRAD Pq, Ib	invalid	PSLLD Pq, Ib	invalid
		66	invalid	invalid	PSRLD Vdq, Ib	invalid	PSRAD Vdq, Ib	invalid	PSLLD Vdq, Ib	invalid
		F2, F3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 14	0F 73	none	invalid	invalid	PSRLQ Pq, Ib	invalid	invalid	invalid	PSLLQ Pq, Ib	invalid
		66	invalid	invalid	PSRLQ Vdq, Ib	PSRLDQ Vdq, Ib	invalid	invalid	PSLLQ Vdq, Ib	PSLLDQ Vdq, Ib
		F2, F3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid
Group 15	0F AE	none	FXSAVE	FXRSTOR	LDMXCSR	STMXCSR	invalid	LFENCE	MFENCE	SFENCE ³ CLFLUSH Mb
		66, F2, F3	invalid	invalid	invalid	invalid	invalid	invalid	invalid	invalid

Note:

1. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SWAPGS.
2. Invalid in 64-bit mode.
3. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SFENCE.
4. This instruction takes a ModRM byte.
5. Reserved prefetch encodings are aliased to the /0 encoding (PREFETCH Exclusive) if otherwise undefined.

Table 3-6. One-Byte and Two-Byte Opcode ModRM Extensions (continued)

Group Number	Opcode	Prefix	ModRM <i>reg</i> Field							
			/0	/1	/2	/3	/4	/5	/6	/7
Group 16	0F 18	n.a.	PREFETCH NTA	PREFETCH T0	PREFETCH T1	PREFETCH T2	NOP ⁴	NOP ⁴	NOP ⁴	NOP ⁴
Group P	0F 0D	n.a.	PREFETCH Exclusive	PREFETCH Modified	Prefetch Reserved ⁵	PREFETCH Modified	Prefetch Reserved ⁵	Prefetch Reserved ⁵	Prefetch Reserved ⁵	Prefetch Reserved ⁵

Note:

1. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SWAPGS.
2. Invalid in 64-bit mode.
3. See Table 3-7 on page 1141 for ModRM extensions of this two-byte opcode to encode SFENCE.
4. This instruction takes a ModRM byte.
5. Reserved prefetch encodings are aliased to the /0 encoding (PREFETCH Exclusive) if otherwise undefined.

3.2.5 ModRM Extensions to SWAPGS and CLFLUSH Opcodes

Table 3-7 shows the ModRM *r/m* field encodings for the 0F 01 and 0F AE opcodes, shown in Table 3-6. These opcodes are shared by INVLPG and SWAPGS (0F 01) and SFENCE and CLFLUSH (0F AE), respectively. The opcodes are differentiated by the fact that the binary value of the ModRM *mod* field is always 11 for SWAPGS and CLFLUSH, and any value except 11 for INVLPG and SFENCE. The SWAPGS opcode is only valid in 64-bit mode.

Table 3-7. SWAPGS and CLFLUSH ModRM Extensions

Opcode	ModRM <i>r/m</i> Field							
	/0	/1	/2	/3	/4	/5	/6	/7
0F 01	SWAPGS	invalid	invalid	invalid	invalid	invalid	invalid	invalid
0F AE	SFENCE	invalid	invalid	invalid	invalid	invalid	invalid	invalid

3.2.6 3DNow!™ Opcodes

The 64-bit media instructions include the MMX™ instructions and the AMD 3DNow! instructions. The MMX instructions are encoded using two opcode bytes, as described in Section 3.2.2, “Two-Byte Opcodes”, on page 1128.

The 3DNow! instructions are encoded using two 0Fh opcode bytes and an immediate byte that is located at the last byte position of the instruction encoding. Thus, the format for 3DNow! instructions is:

0Fh 0Fh [ModRM] [SIB] [displacement] *imm8_opcode*

Table 3-8 and Table 3-9 show the immediate byte following the opcode bytes for 3DNow! instructions. In these tables, rows show the high nibble of the immediate byte, and columns show the low nibble of the immediate byte. Table 3-8 shows the immediate bytes whose low nibble is in the range 0:7h. Table 3-9 shows the same for immediate bytes whose low nibble is in the range 8:Fh.

Byte values shown as *reserved* in these tables have implementation-specific functions, which can include an invalid-opcode exception.

Table 3-8. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 0:7h

Nibble ¹	0	1	2	3	4	5	6	7
0	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
2	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
3	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
4	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
5	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
6	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
7	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
8	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
9	PFCMPGE Pq, Qq	reserved	reserved	reserved	PFCMPGE Pq, Qq	reserved	PFCMPGE Pq, Qq	PFCMPGE Pq, Qq
A	PFCMPGT Pq, Qq	reserved	reserved	reserved	PFCMPGT Pq, Qq	reserved	PFCMPGT Pq, Qq	PFCMPGT Pq, Qq
B	PFCMPEQ Pq, Qq	reserved	reserved	reserved	PFCMPEQ Pq, Qq	reserved	PFCMPEQ Pq, Qq	PFCMPEQ Pq, Qq
C	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Note:

1. All 3DNow! opcodes consist of two 0Fh bytes. This table shows the immediate byte for 3DNow! opcodes. Rows show the high nibble of the immediate byte. Columns show the low nibble of the immediate byte.

Table 3-8. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 0:7h (continued)

Nibble ¹	0	1	2	3	4	5	6	7
D	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
E	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
F	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Note:

1. All 3DNow! opcodes consist of two 0Fh bytes. This table shows the immediate byte for 3DNow! opcodes. Rows show the high nibble of the immediate byte. Columns show the low nibble of the immediate byte.

Table 3-9. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 8:Fh

Nibble ¹	8	9	A	B	C	D	E	F
0	reserved	reserved	reserved	reserved	PI2FW Pq, Qq	PI2FD Pq, Qq	reserved	reserved
2	reserved	reserved	reserved	reserved	PF2IW Pq, Qq	PF2ID Pq, Qq	reserved	reserved
3	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
4	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
5	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
6	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
7	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
8	reserved	reserved	PFNACC Pq, Qq	reserved	reserved	reserved	PFNACC Pq, Qq	reserved
9	reserved	reserved	PFSUB Pq, Qq	reserved	reserved	reserved	PFADD Pq, Qq	reserved
A	reserved	reserved	PFSUBR Pq, Qq	reserved	reserved	reserved	PFACC Pq, Qq	reserved
B	reserved	reserved	reserved	PSWAPD Pq, Qq	reserved	reserved	reserved	PAVGUSB Pq, Qq
C	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Note:

1. All 3DNow! opcodes consist of two 0Fh bytes. This table shows the immediate byte for 3DNow! opcodes. Rows show the high nibble of the immediate byte. Columns show the low nibble of the immediate byte.

Table 3-9. Immediate Byte for 3DNow!™ Opcodes, Low Nibble 8:Fh (continued)

Nibble ¹	8	9	A	B	C	D	E	F
D	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
E	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved
F	reserved	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Note:

1. All 3DNow! opcodes consist of two 0Fh bytes. This table shows the immediate byte for 3DNow! opcodes. Rows show the high nibble of the immediate byte. Columns show the low nibble of the immediate byte.

3.2.7 x87 Encodings

All x87 instructions begin with an opcode byte in the range D8h to DFh, as shown in Table 3-2 on page 1127. These opcodes are followed by a ModRM byte which further defines the opcode. Table 3-10 shows both the opcode byte and the ModRM byte for each x87 instruction.

There are two significant ranges of the ModRM *reg* field (bits 5:3) for x87 opcodes: 00:BFh and C0:FFh. When the value of the ModRM *reg* field falls within the first range, 00:BFh, the opcode uses only this *reg* field to further define the opcode. When the value of the *reg* field falls within the second range, C0:FFh, the opcode uses the entire ModRM byte to further define the opcode.

Byte values shown as *reserved* in Table 3-10 have implementation-specific functions, which can include an invalid-opcode exception.

Table 3-10. x87 Opcodes and ModRM Extensions

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
D8	!11	00:BF							
		FADD mem32real	FMUL mem32real	FCOM mem32real	FCOMP mem32real	FSUB mem32real	FSUBR mem32real	FDIV mem32real	FDIVR mem32real
	11	C0 FADD ST(0), ST(0)	C8 FMUL ST(0), ST(0)	D0 FCOM ST(0), ST(0)	D8 FCOMP ST(0), ST(0)	E0 FSUB ST(0), ST(0)	E8 FSUBR ST(0), ST(0)	F0 FDIV ST(0), ST(0)	F8 FDIVR ST(0), ST(0)
		C1 FADD ST(0), ST(1)	C9 FMUL ST(0), ST(1)	D1 FCOM ST(0), ST(1)	D9 FCOMP ST(0), ST(1)	E1 FSUB ST(0), ST(1)	E9 FSUBR ST(0), ST(1)	F1 FDIV ST(0), ST(1)	F9 FDIVR ST(0), ST(1)
		C2 FADD ST(0), ST(2)	CA FMUL ST(0), ST(2)	D2 FCOM ST(0), ST(2)	DA FCOMP ST(0), ST(2)	E2 FSUB ST(0), ST(2)	EA FSUBR ST(0), ST(2)	F2 FDIV ST(0), ST(2)	FA FDIVR ST(0), ST(2)
		C3 FADD ST(0), ST(3)	CB FMUL ST(0), ST(3)	D3 FCOM ST(0), ST(3)	DB FCOMP ST(0), ST(3)	E3 FSUB ST(0), ST(3)	EB FSUBR ST(0), ST(3)	F3 FDIV ST(0), ST(3)	FB FDIVR ST(0), ST(3)
		C4 FADD ST(0), ST(4)	CC FMUL ST(0), ST(4)	D4 FCOM ST(0), ST(4)	DC FCOMP ST(0), ST(4)	E4 FSUB ST(0), ST(4)	EC FSUBR ST(0), ST(4)	F4 FDIV ST(0), ST(4)	FC FDIVR ST(0), ST(4)
		C5 FADD ST(0), ST(5)	CD FMUL ST(0), ST(5)	D5 FCOM ST(0), ST(5)	DD FCOMP ST(0), ST(5)	E5 FSUB ST(0), ST(5)	ED FSUBR ST(0), ST(5)	F5 FDIV ST(0), ST(5)	FD FDIVR ST(0), ST(5)
		C6 FADD ST(0), ST(6)	CE FMUL ST(0), ST(6)	D6 FCOM ST(0), ST(6)	DE FCOMP ST(0), ST(6)	E6 FSUB ST(0), ST(6)	EE FSUBR ST(0), ST(6)	F6 FDIV ST(0), ST(6)	FE FDIVR ST(0), ST(6)
		C7 FADD ST(0), ST(7)	CF FMUL ST(0), ST(7)	D7 FCOM ST(0), ST(7)	DF FCOMP ST(0), ST(7)	E7 FSUB ST(0), ST(7)	EF FSUBR ST(0), ST(7)	F7 FDIV ST(0), ST(7)	FF FDIVR ST(0), ST(7)

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
D9	!11	00:BF							
		FLD mem32real	invalid	FST mem32real	FSTP mem32real	FLDENV mem14/28e nv	FLDCW mem16	FSTENV mem14/28e nv	FSTCW mem16
	11	C0 FLD ST(0), ST(0)	C8 FXCH ST(0), ST(0)	D0 FNOP	D8 invalid	E0 FCHS	E8 FLD1	F0 F2XM1	F8 FPREM
		C1 FLD ST(0), ST(1)	C9 FXCH ST(0), ST(1)	D1 invalid	D9 invalid	E1 FABS	E9 FLDL2T	F1 FYL2X	F9 FYL2XP1
		C2 FLD ST(0), ST(2)	CA FXCH ST(0), ST(2)	D2 invalid	DA invalid	E2 invalid	EA FLDL2E	F2 FPTAN	FA FSQRT
		C3 FLD ST(0), ST(3)	CB FXCH ST(0), ST(3)	D3 invalid	DB invalid	E3 invalid	EB FLDPI	F3 FPATAN	FB FSINCOS
		C4 FLD ST(0), ST(4)	CC FXCH ST(0), ST(4)	D4 invalid	DC invalid	E4 FTST	EC FLDLG2	F4 EXTRACT	FC FRNDINT
		C5 FLD ST(0), ST(5)	CD FXCH ST(0), ST(5)	D5 invalid	DD invalid	E5 FXAM	ED FLDLN2	F5 FPREM1	FD FSCALE
		C6 FLD ST(0), ST(6)	CE FXCH ST(0), ST(6)	D6 invalid	DE invalid	E6 invalid	EE FLDZ	F6 FDECSTP	FE FSIN
		C7 FLD ST(0), ST(7)	CF FXCH ST(0), ST(7)	D7 invalid	DF invalid	E7 invalid	EF invalid	F7 FINCSTP	FF FCOS

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DA	!11	00:BF							
		FIADD mem32int	FIMUL mem32int	FICOM mem32int	FICOMP mem32int	FISUB mem32int	FISUBR mem32int	FIDIV mem32int	FIDIVR mem32int
	11	C0 FCMOVB ST(0), ST(0)	C8 FCMOVE ST(0), ST(0)	D0 FCMOVBE ST(0), ST(0)	D8 FCMOVU ST(0), ST(0)	E0 invalid	E8 invalid	F0 invalid	F8 invalid
		C1 FCMOVB ST(0), ST(1)	C9 FCMOVE ST(0), ST(1)	D1 FCMOVBE ST(0), ST(1)	D9 FCMOVU ST(0), ST(1)	E1 invalid	E9 FUCOMPP	F1 invalid	F9 invalid
		C2 FCMOVB ST(0), ST(2)	CA FCMOVE ST(0), ST(2)	D2 FCMOVBE ST(0), ST(2)	DA FCMOVU ST(0), ST(2)	E2 invalid	EA invalid	F2 invalid	FA invalid
		C3 FCMOVB ST(0), ST(3)	CB FCMOVE ST(0), ST(3)	D3 FCMOVBE ST(0), ST(3)	DB FCMOVU ST(0), ST(3)	E3 invalid	EB invalid	F3 invalid	FB invalid
		C4 FCMOVB ST(0), ST(4)	CC FCMOVE ST(0), ST(4)	D4 FCMOVBE ST(0), ST(4)	DC FCMOVU ST(0), ST(4)	E4 invalid	EC invalid	F4 invalid	FC invalid
		C5 FCMOVB ST(0), ST(5)	CD FCMOVE ST(0), ST(5)	D5 FCMOVBE ST(0), ST(5)	DD FCMOVU ST(0), ST(5)	E5 invalid	ED invalid	F5 invalid	FD invalid
		C6 FCMOVB ST(0), ST(6)	CE FCMOVE ST(0), ST(6)	D6 FCMOVBE ST(0), ST(6)	DE FCMOVU ST(0), ST(6)	E6 invalid	EE invalid	F6 invalid	FE invalid
		C7 FCMOVB ST(0), ST(7)	CF FCMOVE ST(0), ST(7)	D7 FCMOVBE ST(0), ST(7)	DF FCMOVU ST(0), ST(7)	E7 invalid	EF invalid	F7 invalid	FF invalid

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DB	!11	00:BF							
		FILD mem32int	invalid	FIST mem32int	FISTP mem32int	invalid	FLD mem80real	invalid	FSTP mem80real
	11	C0 FCMOVNB ST(0), ST(0)	C8 FCMOVNE ST(0), ST(0)	D0 FCMOVNBE ST(0), ST(0)	D8 FCMOVNU ST(0), ST(0)	E0 invalid	E8 FUCOMI ST(0), ST(0)	F0 FCOMI ST(0), ST(0)	F8 invalid
		C1 FCMOVNB ST(0), ST(1)	C9 FCMOVNE ST(0), ST(1)	D1 FCMOVNBE ST(0), ST(1)	D9 FCMOVNU ST(0), ST(1)	E1 invalid	E9 FUCOMI ST(0), ST(1)	F1 FCOMI ST(0), ST(1)	F9 invalid
		C2 FCMOVNB ST(0), ST(2)	CA FCMOVNE ST(0), ST(2)	D2 FCMOVNBE ST(0), ST(2)	DA FCMOVNU ST(0), ST(2)	E2 FCLEX	EA FUCOMI ST(0), ST(2)	F2 FCOMI ST(0), ST(2)	FA invalid
		C3 FCMOVNB ST(0), ST(3)	CB FCMOVNE ST(0), ST(3)	D3 FCMOVNBE ST(0), ST(3)	DB FCMOVNU ST(0), ST(3)	E3 FINIT	EB FUCOMI ST(0), ST(3)	F3 FCOMI ST(0), ST(3)	FB invalid
		C4 FCMOVNB ST(0), ST(4)	CC FCMOVNE ST(0), ST(4)	D4 FCMOVNBE ST(0), ST(4)	DC FCMOVNU ST(0), ST(4)	E4 invalid	EC FUCOMI ST(0), ST(4)	F4 FCOMI ST(0), ST(4)	FC invalid
		C5 FCMOVNB ST(0), ST(5)	CD FCMOVNE ST(0), ST(5)	D5 FCMOVNBE ST(0), ST(5)	DD FCMOVNU ST(0), ST(5)	E5 invalid	ED FUCOMI ST(0), ST(5)	F5 FCOMI ST(0), ST(5)	FD invalid
		C6 FCMOVNB ST(0), ST(6)	CE FCMOVNE ST(0), ST(6)	D6 FCMOVNBE ST(0), ST(6)	DE FCMOVNU ST(0), ST(6)	E6 invalid	EE FUCOMI ST(0), ST(6)	F6 FCOMI ST(0), ST(6)	FE invalid
		C7 FCMOVNB ST(0), ST(7)	CF FCMOVNE ST(0), ST(7)	D7 FCMOVNBE ST(0), ST(7)	DF FCMOVNU ST(0), ST(7)	E7 invalid	EF FUCOMI ST(0), ST(7)	F7 FCOMI ST(0), ST(7)	FF invalid

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DC	!11	00:BF							
		FADD m64real	FMUL m64real	FCOM m64real	FCOMP m64real	FSUB m64real	FSUBR m64real	FDIV m64real	FDIVR m64real
	11	C0 FADD ST(0), ST(0)	C8 FMUL ST(0), ST(0)	D0 invalid	D8 invalid	E0 FSUBR ST(0), ST(0)	E8 FSUB ST(0), ST(0)	F0 FDIVR ST(0), ST(0)	F8 FDIV ST(0), ST(0)
		C1 FADD ST(1), ST(0)	C9 FMUL ST(1), ST(0)	D1 FADD ST(1), ST(0)	D9 FADD ST(1), ST(0)	E1 FSUBR ST(1), ST(0)	E9 FSUB ST(1), ST(0)	F1 FDIVR ST(1), ST(0)	F9 FDIV ST(1), ST(0)
		C2 FADD ST(2), ST(0)	CA FMUL ST(2), ST(0)	D2 invalid	DA invalid	E2 FSUBR ST(2), ST(0)	EA FSUB ST(2), ST(0)	F2 FDIVR ST(2), ST(0)	FA FDIV ST(2), ST(0)
		C3 FADD ST(3), ST(0)	CB FMUL ST(3), ST(0)	D3 invalid	DB invalid	E3 FSUBR ST(3), ST(0)	EB FSUB ST(3), ST(0)	F3 FDIVR ST(3), ST(0)	FB FDIV ST(3), ST(0)
		C4 FADD ST(4), ST(0)	CC FMUL ST(4), ST(0)	D4 invalid	DC invalid	E4 FSUBR ST(4), ST(0)	EC FSUB ST(4), ST(0)	F4 FDIVR ST(4), ST(0)	FC FDIV ST(4), ST(0)
		C5 FADD ST(5), ST(0)	CD FMUL ST(5), ST(0)	D5 invalid	DD invalid	E5 FSUBR ST(5), ST(0)	ED FSUB ST(5), ST(0)	F5 FDIVR ST(5), ST(0)	FD FDIV ST(5), ST(0)
		C6 FADD ST(6), ST(0)	CE FMUL ST(6), ST(0)	D6 invalid	DE invalid	E6 FSUBR ST(6), ST(0)	EE FSUB ST(6), ST(0)	F6 FDIVR ST(6), ST(0)	FE FDIV ST(6), ST(0)
		C7 FADD ST(7), ST(0)	CF FMUL ST(7), ST(0)	D7 invalid	DF invalid	E7 FSUBR ST(7), ST(0)	EF FSUB ST(7), ST(0)	F7 FDIVR ST(7), ST(0)	FF FDIV ST(7), ST(0)

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DD	!11	00:BF							
		FLD m64real	invalid	FST m64real	FSTP m64real	FRSTOR mem98/108 env	invalid	FSAVE mem98/108 env	FSTSW mem16
	11	C0 FFREE ST(0)	C8 invalid	D0 FST ST(0)	D8 FSTP ST(0)	E0 FUCOM ST(0), ST(0)	E8 FUCOMP ST(0)	F0 invalid	F8 invalid
		C1 FFREE ST(1)	C9 invalid	D1 FST ST(1)	D9 FSTP ST(1)	E1 FUCOM ST(1), ST(0)	E9 FUCOMP ST(1)	F1 invalid	F9 invalid
		C2 FFREE ST(2)	CA invalid	D2 FST ST(2)	DA FSTP ST(2)	E2 FUCOM ST(2), ST(0)	EA FUCOMP ST(2)	F2 invalid	FA invalid
		C3 FFREE ST(3)	CB invalid	D3 FST ST(3)	DB FSTP ST(3)	E3 FUCOM ST(3), ST(0)	EB FUCOMP ST(3)	F3 invalid	FB invalid
		C4 FFREE ST(4)	CC invalid	D4 FST ST(4)	DC FSTP ST(4)	E4 FUCOM ST(4), ST(0)	EC FUCOMP ST(4)	F4 invalid	FC invalid
		C5 FFREE ST(5)	CD invalid	D5 FST ST(5)	DD FSTP ST(5)	E5 FUCOM ST(5), ST(0)	ED FUCOMP ST(5)	F5 invalid	FD invalid
		C6 FFREE ST(6)	CE invalid	D6 FST ST(6)	DE FSTP ST(6)	E6 FUCOM ST(6), ST(0)	EE FUCOMP ST(6)	F6 invalid	FE invalid
		C7 FFREE ST(7)	CF invalid	D7 FST ST(7)	DF FSTP ST(7)	E7 FUCOM ST(7), ST(0)	EF FUCOMP ST(7)	F7 invalid	FF invalid

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DE	!11	00:BF							
		FIADD mem16int	FIMUL mem16int	FICOM mem16int	FICOMP mem16int	FISUB mem16int	FISUBR mem16int	FIDIV mem16int	FIDIVR mem16int
	11	C0 FADDP ST(0), ST(0)	C8 FMULP ST(0), ST(0)	D0 invalid	D8 invalid	E0 FSUBRP ST(0), ST(0)	E8 FSUBP ST(0), ST(0)	F0 FDIVRP ST(0), ST(0)	F8 FDIVP ST(0), ST(0)
		C1 FADDP ST(1), ST(0)	C9 FMULP ST(1), ST(0)	D1 invalid	D9 FCOMPP	E1 FSUBRP ST(1), ST(0)	E9 FSUBP ST(1), ST(0)	F1 FDIVRP ST(1), ST(0)	F9 FDIVP ST(1), ST(0)
		C2 FADDP ST(2), ST(0)	CA FMULP ST(2), ST(0)	D2 invalid	DA invalid	E2 FSUBRP ST(2), ST(0)	EA FSUBP ST(2), ST(0)	F2 FDIVRP ST(2), ST(0)	FA FDIVP ST(2), ST(0)
		C3 FADDP ST(3), ST(0)	CB FMULP ST(3), ST(0)	D3 invalid	DB invalid	E3 FSUBRP ST(3), ST(0)	EB FSUBP ST(3), ST(0)	F3 FDIVRP ST(3), ST(0)	FB FDIVP ST(3), ST(0)
		C4 FADDP ST(4), ST(0)	CC FMULP ST(4), ST(0)	D4 invalid	DC invalid	E4 FSUBRP ST(4), ST(0)	EC FSUBP ST(4), ST(0)	F4 FDIVRP ST(4), ST(0)	FC FDIVP ST(4), ST(0)
		C5 FADDP ST(5), ST(0)	CD FMULP ST(5), ST(0)	D5 invalid	DD invalid	E5 FSUBRP ST(5), ST(0)	ED FSUBP ST(5), ST(0)	F5 FDIVRP ST(5), ST(0)	FD FDIVP ST(5), ST(0)
		C6 FADDP ST(6), ST(0)	CE FMULP ST(6), ST(0)	D6 invalid	DE invalid	E6 FSUBRP ST(6), ST(0)	EE FSUBP ST(6), ST(0)	F6 FDIVRP ST(6), ST(0)	FE FDIVP ST(6), ST(0)
		C7 FADDP ST(7), ST(0)	CF FMULP ST(7), ST(0)	D7 invalid	DF invalid	E7 FSUBRP ST(7), ST(0)	EF FSUBP ST(7), ST(0)	F7 FDIVRP ST(7), ST(0)	FF FDIVP ST(7), ST(0)

Table 3-10. x87 Opcodes and ModRM Extensions (continued)

Opcode	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
DF	!11	00:BF							
		FILD mem16int	invalid	FIST mem16int	FISTP mem16int	FBLD mem80dec	FILD m64int	FBSTP mem80dec	FISTP m64int
	11	C0 invalid	C8 invalid	D0 invalid	D8 invalid	E0 FSTSW AX	E8 FUCOMIP ST(0), ST(0)	F0 FCOMIP ST(0), ST(0)	F8 invalid
		C1 invalid	C9 invalid	D1 invalid	D9 invalid	E1 invalid	E9 FUCOMIP ST(0), ST(1)	F1 FCOMIP ST(0), ST(1)	F9 invalid
		C2 invalid	CA invalid	D2 invalid	DA invalid	E2 invalid	EA FUCOMIP ST(0), ST(2)	F2 FCOMIP ST(0), ST(2)	FA invalid
		C3 invalid	CB invalid	D3 invalid	DB invalid	E3 invalid	EB FUCOMIP ST(0), ST(3)	F3 FCOMIP ST(0), ST(3)	FB invalid
		C4 invalid	CC invalid	D4 invalid	DC invalid	E4 invalid	EC FUCOMIP ST(0), ST(4)	F4 FCOMIP ST(0), ST(4)	FC invalid
		C5 invalid	CD invalid	D5 invalid	DD invalid	E5 invalid	ED FUCOMIP ST(0), ST(5)	F5 FCOMIP ST(0), ST(5)	FD invalid
		C6 invalid	CE invalid	D6 invalid	DE invalid	E6 invalid	EE FUCOMIP ST(0), ST(6)	F6 FCOMIP ST(0), ST(6)	FE invalid
		C7 invalid	CF invalid	D7 invalid	DF invalid	E7 invalid	EF FUCOMIP ST(0), ST(7)	F7 FCOMIP ST(0), ST(7)	FF invalid

3.2.8 rFLAGS Condition Codes for x87 Opcodes

Table 3-11 shows the rFLAGS condition codes specified by the opcode and ModRM bytes of the FCMOV_{cc} instructions.

Table 3-11. rFLAGS Condition Codes for FCMOV_{cc}

Opcode (hex)	ModRM <i>mod</i> Field	ModRM <i>reg</i> Field	rFLAGS Value	cc Mnemonic	Condition
DA	11	000	CF = 1	B	Below
		001	ZF = 1	E	Equal
		010	CF =1 or ZF = 1	BE	Below or Equal
		011	PF = 1	U	Unordered
DB		000	CF = 0	NB	Not Below
		001	ZF = 0	NE	Not Equal
		010	CF = 0 or ZF = 0	NBE	Not Below or Equal
		011	PF = 0	NU	Not Unordered

3.3 Operand Encodings

Register and memory operands are encoded using the *mode-register-memory* (ModRM) and the *scale-index-base* (SIB) bytes that follow the opcodes. In some instructions, the ModRM byte is followed by a SIB byte, which defines the instruction's memory-addressing mode for the complex-addressing modes.

3.3.1 ModRM Operand References

Figure 3-2 shows the format of a ModRM byte. There are three fields—*mod*, *reg*, and *r/m*. The *reg* field not only provides additional opcode bits—as described above in Section 3.2.4 through Section 3.2.7—but is also used with the other two fields to specify operands. The *mod* and *r/m* fields are used together with each other and, in 64-bit mode, with the REX.R and REX.B bits of the REX prefix, to specify the location of the instruction's operands and certain of the possible addressing modes (specifically, the non-complex modes).

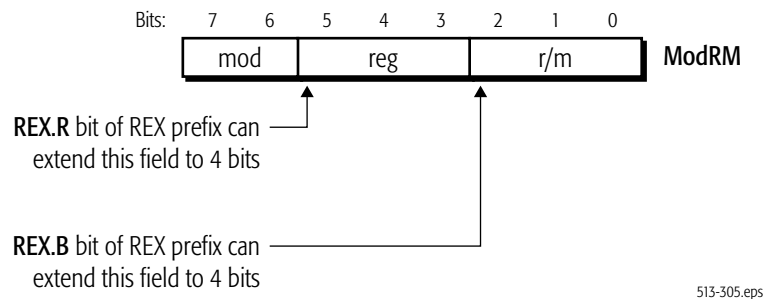


Figure 3-2. ModRM-Byte Format

The two sections below describe the ModRM operand encodings, first for 32-bit and 64-bit references, and then for 16-bit references.

32-Bit and 64-Bit Register and Memory References. Table 3-12 shows the encoding for 32-bit and 64-bit register references using the ModRM *reg* field. The first nine rows of Table 3-12 show references when the REX.R bit is cleared to 0, and the last nine rows show references when the REX.R bit is set to 1. In this table, *Mnemonic Notation* means the syntax notation shown in Section 2.5.1 on page 41 for a register, and *ModRM Notation (/r)* means the opcode-syntax notation shown in Section 2.5.2 on page 44 for the register.

Table 3-13 shows the encoding for 32-bit and 64-bit memory references using the ModRM byte. This table describes 32-bit and 64-bit addressing, with the REX.B bit set or cleared. The *Effective Address* is shown in the two left-most columns, followed by the binary encoding of the ModRM-byte *mod* field, followed by the eight possible hex values of the complete ModRM byte (one value for each binary encoding of the ModRM-byte *reg* field), followed by the binary encoding of the ModRM *r/m* field.

The /0 through /7 notation for the ModRM *reg* field (bits 5:3) means that the three-bit field contains a value from zero (binary 000) to 7 (binary 111).

Table 3-12. ModRM Register References, 32/64-Bit Address

Mnemonic Notation	REX.R Bit	ModRM <i>reg</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
reg8	0	AL	CL	DL	BL	AH/SPL	CH/BPL	DH/SIL	BH/DIL
reg16		AX	CX	DX	BX	SP	BP	SI	DI
reg32		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
reg64		RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
mmx		MMX0	MMX1	MMX2	MMX3	MMX4	MMX5	MMX6	MMX7
xmm		XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7
sReg		ES	CS	SS	DS	FS	GS	reserved	reserved
cReg		CR0	CR1	CR2	CR3	CR4	CR5	CR6	CR7
dReg		DR0	DR1	DR2	DR3	DR4	DR5	DR6	DR7
reg8	1	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B
reg16		R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
reg32		R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
reg64		R8	R9	R10	R11	R12	R13	R14	R15
mmx		MMX0	MMX1	MMX2	MMX3	MMX4	MMX5	MMX6	MMX7
xmm		XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15
sReg		ES	CS	SS	DS	FS	GS	reserved	reserved
cReg		CR8	CR9	CR10	CR11	CR12	CR13	CR14	CR15
dReg		DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15

Table 3-13. ModRM Memory References, 32/64-Bit Address

Effective Address ¹		ModRM mod Field (binary)	ModRM reg Field ³								ModRM r/m Field (binary)
			/0	/1	/2	/3	/4	/5	/6	/7	
REX.B = 0	REX.B = 1		Complete ModRM Byte (hex)								
[rAX]	[r8]	00	00	08	10	18	20	28	30	38	000
[rCX]	[r9]		01	09	11	19	21	29	31	39	001
[rDX]	[r10]		02	0A	12	1A	22	2A	32	3A	010
[rBX]	[r11]		03	0B	13	1B	23	2B	33	3B	011
[sib]	[sib]		04	0C	14	1C	24	2C	34	3C	100
[RIP+disp32] or [disp32] ²	[rIP+disp32] or [disp32] ²		05	0D	15	1D	25	2D	35	3D	101
[rSI]	[r14]		06	0E	16	1E	26	2E	36	3E	110
[rDI]	[r15]		07	0F	17	1F	27	2F	37	3F	111
[rAX+disp8]	[r8+disp8]	01	40	48	50	58	60	68	70	78	000
[rCX+disp8]	[r9+disp8]		41	49	51	59	61	69	71	79	001
[rDX+disp8]	[r10+disp8]		42	4A	52	5A	62	6A	72	7A	010
[rBX+disp8]	[r11+disp8]		43	4B	53	5B	63	6B	73	7B	011
[SIB-byte+disp8]	[SIB-byte+disp8]		44	4C	54	5C	64	6C	74	7C	100
[rBP+disp8]	[r13+disp8]		45	4D	55	5D	65	6D	75	7D	101
[rSI+disp8]	[r14+disp8]		46	4E	56	5E	66	6E	76	7E	110
[rDI+disp8]	[r15+disp8]		47	4F	57	5F	67	6F	77	7F	111
[rAX+disp32]	[r8+disp32]	10	80	88	90	98	A0	A8	B0	B8	000
[rCX+disp32]	[r9+disp32]		81	89	91	99	A1	A9	B1	B9	001
[rDX+disp32]	[r10+disp32]		82	8A	92	9A	A2	AA	B2	BA	010
[rBX+disp32]	[r11+disp32]		83	8B	93	9B	A3	AB	B3	BB	011
[sib+disp32]	[sib+disp32]		84	8C	94	9C	A4	AC	B4	BC	100
[rBP+disp32]	[r13+disp32]		85	8D	95	9D	A5	AD	B5	BD	101
[rSI+disp32]	[r14+disp32]		86	8E	96	9E	A6	AE	B6	BE	110
[rDI+disp32]	[r15+disp32]		87	8F	97	9F	A7	AF	B7	BF	111

Note:

1. "disp8" and "disp32" indicate an 8-bit or 32-bit displacement.
2. In 64-bit mode, the effective address is [RIP+disp32]. In compatibility mode, the effective address is [disp32]. If the address-size prefix is used in 64-bit mode to override 64-bit addressing, the [RIP+disp32] effective address is truncated after computation to 32 bits.
3. See Table 3-12 for complete specification of ModRM "reg" field.

Table 3-13. ModRM Memory References, 32/64-Bit Address (continued)

Effective Address ¹		ModRM mod Field (binary)	ModRM reg Field ³								ModRM r/m Field (binary)
			/0	/1	/2	/3	/4	/5	/6	/7	
REX.B = 0	REX.B = 1		Complete ModRM Byte (hex)								
AL/rAX/MMX0/XMM0	r8/MMX0/XMM8	11	C0	C8	D0	D8	E0	E8	F0	F8	000
CL/rCX/MMX1/XMM1	r9/MMX1/XMM9		C1	C9	D1	D9	E1	E9	F1	F9	001
DL/rDX/MMX2/XMM2	r10/MMX2/XMM10		C2	CA	D2	DA	E2	EA	F2	FA	010
BL/rBX/MMX3/XMM3	r11/MMX3/XMM11		C3	CB	D3	DB	E3	EB	F3	FB	011
AH/SPL/rSP/MMX4/XMM4	r12/MMX4/XMM12		C4	CC	D4	DC	E4	EC	F4	FC	100
CH/BPL/rBP/MMX5/XMM5	r13/MMX5/XMM13		C5	CD	D5	DD	E5	ED	F5	FD	101
DH/SIL/rSI/MMX6/XMM6	r14/MMX6/XMM14		C6	CE	D6	DE	E6	EE	F6	FE	110
BHDIL/rDI/MMX7/XMM7	r15/MMX7/XMM15		C7	CF	D7	DF	E7	EF	F7	FF	111
Note: 1. “disp8” and “disp32” indicate an 8-bit or 32-bit displacement. 2. In 64-bit mode, the effective address is [RIP+disp32]. In compatibility mode, the effective address is [disp32]. If the address-size prefix is used in 64-bit mode to override 64-bit addressing, the [RIP+disp32] effective address is truncated after computation to 32 bits. 3. See Table 3-12 for complete specification of ModRM “reg” field.											

16-Bit Register and Memory References. Table 3-14 shows the notation and encoding conventions for 16-bit register references using the ModRM reg field. This table is comparable to Table 3-12 but applies only to 16-bit operands (the 66h 16-bit operand-size prefix cannot be used with a REX operand-size prefix). Table 3-15 shows the notation and encoding conventions for 16-bit memory references using the ModRM byte. This table is comparable to Table 3-13.

Table 3-14. ModRM Register References, 16-Bit Address

Mnemonic Notation	ModRM reg Field							
	/0	/1	/2	/3	/4	/5	/6	/7
reg8	AL	CL	DL	BL	AH	CH	DH	BH
reg16	AX	CX	DX	BX	SP	BP	SI	DI
reg32	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
mmx	MMX0	MMX1	MMX2	MMX3	MMX4	MMX5	MMX6	MMX7
xmm	XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7

Table 3-14. ModRM Register References, 16-Bit Address (continued)

Mnemonic Notation	ModRM <i>reg</i> Field							
	/0	/1	/2	/3	/4	/5	/6	/7
sReg	ES	CS	SS	DS	FS	GS	reserved	reserved
cReg	CR0	CR1	CR2	CR3	CR4	CR5	CR6	CR7
dReg	DR0	DR1	DR2	DR3	DR4	DR5	DR6	DR7

Table 3-15. ModRM Memory References, 16-Bit Address

Effective Address ¹	ModRM mod Field (binary)	ModRM reg Field ²								ModRM r/m Field (binary)
		/0	/1	/2	/3	/4	/5	/6	/7	
		Complete ModRM Byte (hex)								
[BX+SI]	00	00	08	10	18	20	28	30	38	000
[BX+DI]		01	09	11	19	21	29	31	39	001
[BP+SI]		02	0A	12	1A	22	2A	32	3A	010
[BP+DI]		03	0B	13	1B	23	2B	33	3B	011
[SI]		04	0C	14	1C	24	2C	34	3C	100
[DI]		05	0D	15	1D	25	2D	35	3D	101
[disp16]		06	0E	16	1E	26	2E	36	3E	110
[BX]		07	0F	17	1F	27	2F	37	3F	111
[BX+SI+disp8]	01	40	48	50	58	60	68	70	78	000
[BX+DI+disp8]		41	49	51	59	61	69	71	79	001
[BP+SI+disp8]		42	4A	52	5A	62	6A	72	7A	010
[BP+DI+disp8]		43	4B	53	5B	63	6B	73	7B	011
[SI+disp8]		44	4C	54	5C	64	6C	74	7C	100
[DI+disp8]		45	4D	55	5D	65	6D	75	7D	101
[BP+disp8]		46	4E	56	5E	66	6E	76	7E	110
[BX+disp8]		47	4F	57	5F	67	6F	77	7F	111

Note:

1. "*disp*8" and "*disp*16" indicate an 8-bit or 16-bit displacement.
2. See Table 3-14 for complete specification of ModRM "*reg*" field.

Table 3-15. ModRM Memory References, 16-Bit Address (continued)

Effective Address ¹	ModRM mod Field (binary)	ModRM <i>reg</i> Field ²								ModRM <i>r/m</i> Field (binary)
		/0	/1	/2	/3	/4	/5	/6	/7	
		Complete ModRM Byte (hex)								
[BX+SI+ <i>disp</i> 16]	10	80	88	90	98	A0	A8	B0	B8	000
[BX+DI+ <i>disp</i> 16]		81	89	91	99	A1	A9	B1	B9	001
[BP+SI+ <i>disp</i> 16]		82	8A	92	9A	A2	AA	B2	BA	010
[BP+DI+ <i>disp</i> 16]		83	8B	93	9B	A3	AB	B3	BB	011
[SI+ <i>disp</i> 16]		84	8C	94	9C	A4	AC	B4	BC	100
[DI+ <i>disp</i> 16]		85	8D	95	9D	A5	AD	B5	BD	101
[BP+ <i>disp</i> 16]		86	8E	96	9E	A6	AE	B6	BE	110
[BX+ <i>disp</i> 16]		87	8F	97	9F	A7	AF	B7	BF	111
AL/AX/EAX/MMX0/XMM0	11	C0	C8	D0	D8	E0	E8	F0	F8	000
CL/CX/ECX/MMX1/XMM1		C1	C9	D1	D9	E1	E9	F1	F9	001
DL/DX/EDX/MMX2/XMM2		C2	CA	D2	DA	E2	EA	F2	FA	010
BL/BX/EBX/MMX3/XMM3		C3	CB	D3	DB	E3	EB	F3	FB	011
AH/SP/ESP/MMX4/XMM4		C4	CC	D4	DC	E4	EC	F4	FC	100
CH/BP/EBP/MMX5/XMM5		C5	CD	D5	DD	E5	ED	F5	FD	101
DH/SI/ESI/MMX6/XMM6		C6	CE	D6	DE	E6	EE	F6	FE	110
BH/DI/EDI/MMX7/XMM7		C7	CF	D7	DF	E7	EF	F7	FF	111
Note: 1. “ <i>disp</i> 8” and “ <i>disp</i> 16” indicate an 8-bit or 16-bit displacement. 2. See Table 3-14 for complete specification of ModRM “ <i>reg</i> ” field.										

3.3.2 SIB Operand References

Figure 3-3 shows the format of a scale-index-base (SIB) byte. Some instructions have a SIB byte following their ModRM byte to define memory addressing for the complex-addressing modes described in Section 2.2.3, “Effective Addresses”, on page 17 of Volume 1. The SIB byte has three fields—*scale*, *index*, and *base*—that define the scale factor, index-register number, and base-register number for 32-bit and 64-bit complex addressing modes. In 64-bit mode, the REX.B and REX.X bits extend the encoding of the SIB byte’s *base* and *index* fields.

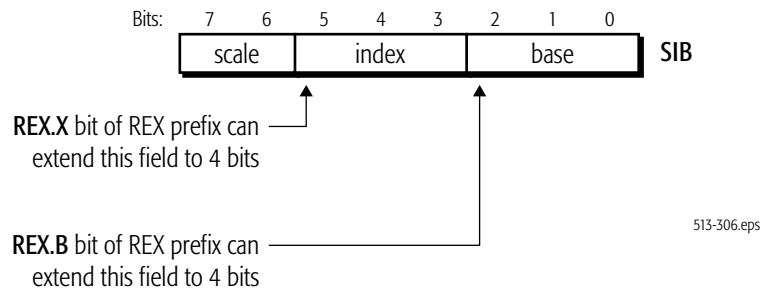
**Figure 3-3. SIB-Byte Format**

Table 3-16 shows the encodings for the SIB byte's *base* field, which specifies the base register for addressing. Table 3-17 shows the encodings for the effective address referenced by a complete SIB byte, including its *scale* and *index* fields. The /0 through /7 notation for the SIB *base* field means that the three-bit field contains a value between zero (binary 000) and 7 (binary 111).

Table 3-16. SIB *base* Field References

REX.B Bit	ModRM <i>mod</i> Field	SIB <i>base</i> Field							
		/0	/1	/2	/3	/4	/5	/6	/7
0	00	rAX	rCX	rDX	rBX	rSP	<i>disp32</i>	rSI	rDI
	01						<i>rBP+disp8</i>		
	10						<i>rBP+disp32</i>		
1	00	r8	r9	r10	r11	r12	<i>disp32</i>	r14	r15
	01						<i>r13+disp8</i>		
	10						<i>r13+disp32</i>		

Table 3-17. SIB Memory References

Effective Address		SIB scale Field	SIB index Field	SIB base Field ¹									
				REX.B = 0:	rAX	rCX	rDX	rBX	rSP	note ¹	rSI	rDI	
				REX.B = 1:	r8	r9	r10	r11	r12	note ¹	r14	r15	
					/0	/1	/2	/3	/4	/5	/6	/7	
REX.X = 0	REX.X = 1			Complete SIB Byte (hex)									
[rAX+base]	[r8+base]	00	000		00	01	02	03	04	05	06	07	
[rCX+base]	[r9+base]		001		08	09	0A	0B	0C	0D	0E	0F	
[rDX+base]	[r10+base]		010		10	11	12	13	14	15	16	17	
[rBX+base]	[r11+base]		011		18	19	1A	1B	1C	1D	1E	1F	
[base]	[r12+base]		100		20	21	22	23	24	25	26	27	
[rBP+base]	[r13+base]		101		28	29	2A	2B	2C	2D	2E	2F	
[rSI+base]	[r14+base]		110		30	31	32	33	34	35	36	37	
[rDI+base]	[r15+base]		111		38	39	3A	3B	3C	3D	3E	3F	
[rAX*2+base]	[r8*2+base]	01	000		40	41	42	43	44	45	46	47	
[rCX*2+base]	[r9*2+base]		001		48	49	4A	4B	4C	4D	4E	4F	
[rDX*2+base]	[r10*2+base]		010		50	51	52	53	54	55	56	57	
[rBX*2+base]	[r11*2+base]		011		58	59	5A	5B	5C	5D	5E	5F	
[base]	[r12*2+base]		100		60	61	62	63	64	65	66	67	
[rBP*2+base]	[r13*2+base]		101		68	69	6A	6B	6C	6D	6E	6F	
[rSI*2+base]	[r14*2+base]		110		70	71	72	73	74	75	76	77	
[rDI*2+base]	[r15*2+base]		111		78	79	7A	7B	7C	7D	7E	7F	
[rAX*4+base]	[r8*4+base]	10	000		80	81	82	83	84	85	86	87	
[rCX*4+base]	[r9*4+base]		001		88	89	8A	8B	8C	8D	8E	8F	
[rDX*4+base]	[r10*4+base]		010		90	91	92	93	94	95	96	97	
[rBX*4+base]	[r11*4+base]		011		98	99	9A	9B	9C	9D	9E	9F	
[base]	[r12*4+base]		100		A0	A1	A2	A3	A4	A5	A6	A7	
[rBP*4+base]	[r13*4+base]		101		A8	A9	AA	AB	AC	AD	AE	AF	
[rSI*4+base]	[r14*4+base]		110		B0	B1	B2	B3	B4	B5	B6	B7	
[rDI*4+base]	[r15*4+base]		111		B8	B9	BA	BB	BC	BD	BE	BF	
Note:													
1. See Table 3-16 on page 1160 for complete specification of SIB “base” field.													

Table 3-17. SIB Memory References (continued)

Effective Address		SIB <i>scale</i> Field	SIB <i>index</i> Field	SIB <i>base</i> Field ¹								
				REX.B = 0:	rAX	rCX	rDX	rBX	rSP	note ¹	rSI	rDI
				REX.B = 1:	r8	r9	r10	r11	r12	note ¹	r14	r15
					/0	/1	/2	/3	/4	/5	/6	/7
REX.X = 0	REX.X = 1			Complete SIB Byte (hex)								
[rAX*8+base]	[r8*8+base]	11	000		C0	C1	C2	C3	C4	C5	C6	C7
[rCX*8+base]	[r9*8+base]		001		C8	C9	CA	CB	CC	CD	CE	CF
[rDX*8+base]	[r10*8+base]		010		D0	D1	D2	D3	D4	D5	D6	D7
[rBX*8+base]	[r11*8+base]		011		D8	D9	DA	DB	DC	DD	DE	DF
[base]	[r12*8+base]		100		E0	E1	E2	E3	E4	E5	E6	E7
[rBP*8+base]	[r13*8+base]		101		E8	E9	EA	EB	EC	ED	EE	EF
[rSI*8+base]	[r14*8+base]		110		F0	F1	F2	F3	F4	F5	F6	F7
[rDI*8+base]	[r15*8+base]		111		F8	F9	FA	FB	FC	FD	FE	FF
Note:												
1. See Table 3-16 on page 1160 for complete specification of SIB “base” field.												

Appendix A General-Purpose Instructions in 64-Bit Mode

This appendix provides details of the general-purpose instructions in 64-bit mode, and its differences from legacy and compatibility modes. The appendix covers only the general-purpose instructions (those described in Section 2.6, “General-Purpose Instruction Reference”, on page 57). It does not cover the 128-bit media, 64-bit media, or x87 floating-point instructions, because those instructions are not affected by 64-bit mode, other than in the access by such instructions to extended GPR and XMM registers when using a REX prefix.

A.1 General Rules for 64-Bit Mode

In 64-bit mode, the following general rules apply to instructions and their operands:

- **“Promoted to 64 Bit”**: If an instruction’s operand size (16-bit or 32-bit) in legacy and compatibility modes depends on the CS.D bit and the operand-size override prefix, then the operand-size choices in 64-bit mode are extended from 16-bit and 32-bit to include 64 bits (with a REX prefix), or the operand size is fixed at 64 bits. Such instructions are said to be “*Promoted to 64 bits*” in Table A-1. However, byte-operand opcodes of such instructions are not promoted.
- **Byte-Operand Opcodes Not Promoted**: As stated above in “Promoted to 64 Bit”, byte-operand opcodes of promoted instructions are not promoted. Those opcodes continue to operate only on bytes.
- **Fixed Operand Size**: If an instruction’s operand size is fixed in legacy mode (thus, independent of CS.D and prefix overrides), that operand size is usually fixed at the same size in 64-bit mode. For example, CUID operates on 32-bit operands, irrespective of attempts to override the operand size. (There are some exceptions, however. For example, BSWAP.)
- **Default Operand Size**: The default operand size for most instructions is 32 bits, and a REX prefix must be used to change the operand size to 64 bits. However, two groups of instructions default to 64-bit operand size and do not need a

REX prefix: (1) near branches and (2) all instructions, except far branches, that implicitly reference the RSP. See Table A-5 on page 1198 for a list of all instructions that default to 64-bit operand size.

- **Zero-Extension of 32-Bit Results:** Operations on 32-bit operands in 64-bit mode zero-extend the high 32 bits of 64-bit GPR destination registers.
- **No Extension of 8-Bit and 16-Bit Results:** Operations on 8-bit and 16-bit operands in 64-bit mode leave the high 56 or 48 bits, respectively, of 64-bit GPR destination registers unchanged.
- **Shift and Rotate Counts:** When the operand size is 64 bits, shifts and rotates use one additional bit (6 bits total) to specify shift-count or rotate-count, allowing 64-bit shifts and rotates.
- **Immediates:** The maximum size of immediate operands is 32 bits, except that 64-bit immediates can be MOVED into 64-bit GPRs. In 64-bit mode, when the operand size is 64 bits, immediates are sign-extended to 64 bits during use, but their actual size (for value representation) remains a maximum of 32 bits.
- **Displacements:** The maximum size of an address displacement is 32 bits. In 64-bit mode, displacements are sign-extended to 64 bits during use, but their actual size (for value representation) remains a maximum of 32 bits.
- **Undefined High 32 Bits After Mode Change:** The processor does not preserve the upper 32 bits of the 64-bit GPRs across switches from 64-bit mode to compatibility or legacy modes. In compatibility or legacy mode, the upper 32 bits of the GPRs are undefined and not accessible to software.

A.2 Operation and Operand Size in 64-Bit Mode

Table A-1 lists the integer instructions, showing operand size in 64-bit mode and the state of the high 32 bits of destination registers when 32-bit operands are used. Opcodes, such as byte-operand versions of several instructions, that do not appear in Table A-1 are covered by the general rules described in Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163.

Table A-1. Operations and Operands in 64-Bit Mode

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
AAA - ASCII Adjust after Addition	INVALID IN 64-BIT MODE (invalid-opcode exception)			
37				
AAD - ASCII Adjust AX before Division	INVALID IN 64-BIT MODE (invalid-opcode exception)			
D5				
AAM - ASCII Adjust AX after Multiply	INVALID IN 64-BIT MODE (invalid-opcode exception)			
D4				
AAS - ASCII Adjust AL after Subtraction	INVALID IN 64-BIT MODE (invalid-opcode exception)			
3F				
ADC —Add with Carry	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
11				
13				
15				
81 /2				
83 /2				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
ADD —Signed or Unsigned Add	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
01				
03				
05				
81 /0				
83 /0				
AND —Logical AND	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
21				
23				
25				
81 /4				
83 /4				
ARPL - Adjust Requestor Privilege Level	OPCODE USED as MOVSD in 64-BIT MODE			
63				
BOUND - Check Array Against Bounds	INVALID IN 64-BIT MODE (invalid-opcode exception)			
62				
BSF —Bit Scan Forward	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F BC				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
BSR —Bit Scan Reverse	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F BD				
BSWAP —Byte Swap	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F C8				
BT —Bit Test	Promoted to 64 bits.	32 bits	No GPR register results.	
0F A3				
0F BA /4				
BTC —Bit Test and Complement	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F BA /7				
0F BB				
BTR —Bit Test and Reset	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F B3				
0F BA /6				
BTS —Bit Test and Set	Promoted to 64 bits.	32 bits	No GPR register results.	
0F AB				
0F BA /5				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
CALL —Procedure Call Near	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
E8	Promoted to 64 bits.	64 bits	RIP = RIP + 32-bit displacement sign-extended to 64 bits. (16-bit displacement size cannot be encoded.)	
FF /2	Promoted to 64 bits.	64 bits	RIP = 64-bit offset from register or memory.	
CALL —Procedure Call Far	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
9A	INVALID IN 64-BIT MODE (invalid-opcode exception)			
FF /3	Promoted to 64 bits.	32 bits	If selector points to a gate, then RIP = zero-extended 32-bit offset from gate, else RIP = zero-extended 32-bit offset from far pointer referenced in instruction.	If selector points to a gate, then RIP = 64-bit offset from gate, else RIP = zero-extended 32-bit offset from far pointer referenced in instruction.
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³	
CBW, CWDE, CDQE —Convert Byte to Word, Convert Word to Doubleword, Convert Doubleword to Quadword	Promoted to 64 bits.	32 bits (size of destination register)	CWDE: Converts word to doubleword. Zero-extends EAX to RAX.	CDQE (new mnemonic): Converts doubleword to quadword. RAX = sign-extended EAX.	
98					
CDQ	see CWD, CDQ, CQO				
CDQE (new mnemonic)	see CBW, CWDE, CDQE				
CDWE	see CBW, CWDE, CDQE				
CLC —Clear Carry Flag	Same as legacy mode.	Not relevant.	No GPR register results.		
F8					
CLD —Clear Direction Flag	Same as legacy mode.	Not relevant.	No GPR register results.		
FC					
CLFLUSH —Cache Line Invalidate	Same as legacy mode.	Not relevant.	No GPR register results.		
0F AE /7					
CLI —Clear Interrupt Flag	Same as legacy mode.	Not relevant.	No GPR register results.		
FA					
CLTS —Clear Task-Switched Flag in CR0	Same as legacy mode.	Not relevant.	No GPR register results.		
0F 06					
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>					

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
CMC —Complement Carry Flag	Same as legacy mode.	Not relevant.	No GPR register results.	
F5				
CMOVcc —Conditional Move	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits. This occurs even if the condition is false.	
0F 42 through 0F 47				
0F 4C through 0F 4F				
CMP —Compare	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
39				
3B				
3D				
81 /7				
83 /7				
CMPS, CMPSB, CMPSW, CMPSD, CMPSQ —Compare Strings	Promoted to 64 bits.	32 bits	CMPSD: Compare String Doublewords. See footnote ⁵	CMPSQ (new mnemonic): Compare String Quadwords See footnote ⁵
A7				
CMPXCHG —Compare and Exchange	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F B1				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
CMPXCHG8B —Compare and Exchange Eight Bytes	Same as legacy mode.	Operand size fixed at 64 bits.	Zero-extends EDX and EAX to 64 bits.	
0F C7 /1				
CPUID —Processor Identification	Same as legacy mode.	Operand size fixed at 32 bits.	Zero-extends 32-bit register results to 64 bits.	
0F A2				
CQO (new mnemonic)	see CWD, CDQ, CQO			
CWD, CDQ, CQO —Convert Word to Doubleword, Convert Doubleword to Quadword, Convert Quadword to Double Quadword	Promoted to 64 bits.	32 bits (size of destination register)	CDQ: Converts doubleword to quadword. Sign-extends EAX to EDX. Zero-extends EDX to RDX. EAX is unchanged.	CQO (new mnemonic): Converts quadword to double quadword. Sign-extends RAX to RDX. RAX is unchanged.
99				
DAA - Decimal Adjust AL after Addition	INVALID IN 64-BIT MODE (invalid-opcode exception)			
27				
DAS - Decimal Adjust AL after Subtraction	INVALID IN 64-BIT MODE (invalid-opcode exception)			
2F				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
DEC —Decrement by 1	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
FF /1				
48 through 4F	OPCODE USED as REX PREFIX in 64-BIT MODE			
DIV —Unsigned Divide	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	RDX:RAX contain a 64-bit quotient (RAX) and 64-bit remainder (RDX).
F7 /6				
ENTER —Create Procedure Stack Frame	Promoted to 64 bits.	64 bits	Can't encode ⁴	
C8				
HLT —Halt	Same as legacy mode.	Not relevant.	No GPR register results.	
F4				
IDIV —Signed Divide	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	RDX:RAX contain a 64-bit quotient (RAX) and 64-bit remainder (RDX).
F7 /7				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
IMUL - Signed Multiply	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
F7 /5				RDX:RAX = RAX * reg/mem64 (i.e., 128-bit result)
0F AF				reg64 = reg64 * reg/mem64
69				reg64 = reg/mem64 * imm32
6B				reg64 = reg/mem64 * imm8
IN —Input From Port	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
E5				
ED				
INC —Increment by 1	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
FF /0				
40 through 47	OPCODE USED as REX PREFIX in 64-BIT MODE			
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
INS, INSB, INSD, INSW —Input String	Same as legacy mode.	32 bits	INSD: Input String Doublewords. Zero-extends 32-bit register results to 64 bits. See footnote ⁵	
6D				
INT n —Interrupt to Vector	Promoted to 64 bits.	Not relevant.	See Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.	
CD				
INT3 —Interrupt to Debug Vector				
CC				
INTO - Interrupt to Overflow Vector	INVALID IN 64-BIT MODE (invalid-opcode exception)			
CE				
INVD —Invalidate Internal Caches	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 08				
INVLPB —Invalidate TLB Entry	Promoted to 64 bits.	Not relevant.	No GPR register results.	
0F 01 /7				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
IRET, IRETD, IRETQ —Interrupt Return	Promoted to 64 bits.	32 bits	IRETD: Interrupt Return Doubleword. See Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.	IRETQ (new mnemonic): Interrupt Return Quadword. See Section 8.8, “Long-Mode Interrupt Control Transfers”, on page 299 of Volume 2.
CF				
Jcc —Jump Conditional	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
70 through 7F	Promoted to 64 bits.	64 bits	RIP = RIP + 8-bit displacement sign-extended to 64 bits.	
0F 80 through 0F 8F	Promoted to 64 bits.	64 bits	RIP = RIP + 32-bit displacement sign-extended to 64 bits. (16-bit displacement size cannot be encoded.)	
JCXZ, JECXZ, JRCXZ —Jump on CX/ECX/RCX Zero	Promoted to 64 bits.	64 bits	RIP = RIP + 8-bit displacement sign-extended to 64 bits.	
E3			See footnote ⁵	
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
JMP —Jump Near	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
EB	Promoted to 64 bits.	64 bits	RIP = RIP + 8-bit displacement sign-extended to 64 bits.	
E9			RIP = RIP + 32-bit displacement sign-extended to 64 bits. (16-bit displacement size cannot be encoded.)	
FF /4			RIP = 64-bit offset from register or memory.	
JMP —Jump Far	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
EA	INVALID IN 64-BIT MODE (invalid-opcode exception)			
FF /5	Promoted to 64 bits.	32 bits	If selector points to a gate, then RIP = zero-extended 32-bit offset from gate, else RIP = zero-extended 32-bit offset from far pointer referenced in instruction.	If selector points to a gate, then RIP = 64-bit offset from gate, else RIP = zero-extended 32-bit offset from far pointer referenced in instruction.
LAHF - Load Status Flags into AH Register	INVALID IN 64-BIT MODE (invalid-opcode exception)			
9F				
Note:				
1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.				
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.				
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.				
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.				
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.				
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
LAR —Load Access Rights Byte	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F 02				
LDS - Load DS Far Pointer	INVALID IN 64-BIT MODE (invalid-opcode exception)			
C5				
LEA —Load Effective Address	Promoted to 64 bits.	32	Zero-extends 32-bit register results to 64 bits.	
8D				
LEAVE —Delete Procedure Stack Frame	Promoted to 64 bits.	64 bits	Can't encode ⁴	
C9				
LES - Load ES Far Pointer	INVALID IN 64-BIT MODE (invalid-opcode exception)			
C4				
LFENCE —Load Fence	Same as legacy mode.	Not relevant.	No GPR register results.	
0F AE /5				
LFS —Load FS Far Pointer	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F B4				
LGDT —Load Global Descriptor Table Register	Promoted to 64 bits.	Operand size fixed at 8+2 bytes.	No GPR register results. Loads 8-byte base and 2-byte limit.	
0F 01 /2				
LGS —Load GS Far Pointer	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F B5				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
LIDT —Load Interrupt Descriptor Table Register	Promoted to 64 bits.	Operand size fixed at 8+2 bytes.	No GPR register results. Loads 8-byte base and 2-byte limit.	
0F 01 /3				
LLDT —Load Local Descriptor Table Register	Promoted to 64 bits.	Operand size fixed at 16 bits.	No GPR register results. References 64-bit-mode descriptor to load 64-bit base.	
0F 00 /2				
LMSW —Load Machine Status Word	Same as legacy mode.	Operand size fixed at 16 bits.	No GPR register results.	
0F 01 /6				
LODS, LODSB, LODSW, LODSD, LODSQ —Load String	Promoted to 64 bits.	32 bits	LODSD: Load String Doublewords. Zero-extends 32-bit register results to 64 bits. See footnote ⁵	LODSQ (new mnemonic): Load String Quadwords. See footnote ⁵
AD				
LOOP —Loop	Promoted to 64 bits.	64 bits	RIP = RIP + 8-bit displacement sign-extended to 64 bits. See footnote ⁵	
E2				
LOOPZ, LOOPE —Loop if Zero/Equal				
E1				
LOOPNZ, LOOPNE —Loop if Not Zero/Equal				
E0				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
LSL —Load Segment Limit	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F 03				
LSS —Load SS Segment Register	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F B2				
LTR —Load Task Register	Promoted to 64 bits.	Operand size fixed at 16 bits.	No GPR register results. References 64-bit-mode descriptor to load 64-bit base.	
0F 00 /3				
MFENCE —Memory Fence	Same as legacy mode.	Not relevant.	No GPR register results.	
0F AE /6				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
MOV —Move	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	32-bit immediate is sign-extended to 64 bits.
89				
8B				
C7			Zero-extends 32-bit register results to 64 bits. Memory offsets are address-sized and default to 64 bits.	64-bit immediate.
B8 through BF				Memory offsets are address-sized and default to 64 bits.
A1 (moffset)				
A3 (moffset)				
MOV —Move to/from Segment Registers	Same as legacy mode.	32 bits	Zero-extends 32-bit register results to 64 bits.	
8C				
8E				
MOV(2) —Move to/from Control Registers	Promoted to 64 bits.	Operand size fixed at 64 bits.	The high 32 bits of control registers differ in their writability and reserved status. See Section 3, “System Resources”, on page 53 of Volume 2 for details.	
0F 22				
0F 20				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
MOV(3) —Move to/from Debug Registers	Promoted to 64 bits.	Operand size fixed at 64 bits.	The high 32 bits of debug registers differ in their writability and reserved status. See Section 13, “Debug and Performance Resources”, on page 395 of Volume 2 for details.	
0F 21				
0F 23				
MOVD —Move Doubleword or Quadword	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F 6E				
0F 7E				
66 0F 6E			Zero-extends 32-bit register results to 128 bits.	Zero-extends 64-bit register results to 128 bits.
66 0F 7E				
MOVNTI —Move Non-Temporal Doubleword	Promoted to 64 bits.	32 bits	No GPR register results.	
0F C3				
MOVS, MOVSB, MOVSW, MOVSD, MOVSQ —Move String	Promoted to 64 bits.	32 bits	MOVSD: Move String Doublewords. Zero-extends 32-bit register results to 64 bits. See footnote ⁵	MOV SQ (new mnemonic): Move String Quadwords. See footnote ⁵
A5				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
MOVSX —Move with Sign-Extend	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Moves byte to quadword.
0F BE				Moves word to quadword.
0F BF				
MOVSXD —Move with Sign-Extend Doubleword	New instruction, available only in 64-bit mode. (In other modes, this opcode is ARPL instruction.)	32 bits	Zero-extends 32-bit register results to 64 bits.	Sign-extends 32 bit register results to 64 bits.
63				
MOVZX —Move with Zero-Extend	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Moves byte to quadword.
0F B6				Moves word to quadword.
0F B7				
MUL —Multiply Unsigned	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	RDX:RAX=RAX * quadword in register or memory.
F7 /4				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
NEG —Negate Two’s Complement	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
F7 /3				
NOP —No Operation	Same as legacy mode.	Not relevant.	No GPR register results.	
90				
NOT —Negate One’s Complement	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
F7 /2				
OR —Logical OR	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
09				
0B				
0D				
81 /1				
83 /1				
OUT —Output to Port	Same as legacy mode.	32 bits	No GPR register results.	
E7				
EF				
OUTS, OUTSB, OUTSD, OUTSW —Output String	Same as legacy mode.	32 bits	OUTSD: Output String Doublewords. No GPR register results. See footnote ⁵	
6F				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
POP —Pop Stack	Promoted to 64 bits.	64 bits	Can't encode ⁴	
8F /0				
58 through 5F				
POP —Pop (segment register from) Stack	Same as legacy mode.	64 bits	Can't encode ⁴	
0F A1 (POP FS)				
0F A9 (POP GS)				
1F (POP DS)	INVALID IN 64-BIT MODE (invalid-opcode exception)			
07 (POP ES)				
17 (POP SS)				
POPA, POPAD - Pop All to GPR Words or Doublewords	INVALID IN 64-BIT MODE (invalid-opcode exception)			
61				
POPF, POPFD, POPFQ —Pop to rFLAGS Word, Doubleword, or Quadword	Promoted to 64 bits.	64 bits	Can't encode ⁴	POPFQ (new mnemonic): Pops 64 bits off stack, writes low 32 bits into EFLAGS and zero-extends the high 32 bits of RFLAGS.
9D				
PREFETCH —Prefetch L1 Data-Cache Line	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 0D /0				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
PREFETCH <i>level</i> —Prefetch Data to Cache Level <i>level</i>	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 18 /0-3				
PREFETCHW —Prefetch L1 Data-Cache Line for Write	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 0D /1				
PUSH —Push onto Stack	Promoted to 64 bits.	64 bits	Can't encode ⁴	
FF /6				
50 through 57				
6A				
68				
PUSH —Push (segment register) onto Stack	Promoted to 64 bits.	64 bits	Can't encode ⁴	
0F A0 (PUSH FS)				
0F A8 (PUSH GS)				
0E (PUSH CS)	INVALID IN 64-BIT MODE (invalid-opcode exception)			
1E (PUSH DS)				
06 (PUSH ES)				
16 (PUSH SS)				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, "General Rules for 64-Bit Mode", on page 1163 for definitions of "Promoted to 64 bits" and related topics.
2. If "Type of Operation" is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, "General Rules for 64-Bit Mode", on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
PUSHA, PUSHAD - Push All to GPR Words or Doublewords	INVALID IN 64-BIT MODE (invalid-opcode exception)			
60				
PUSHF, PUSHFD, PUSHFQ —Push rFLAGS Word, Doubleword, or Quadword onto Stack	Promoted to 64 bits.	64 bits	Can't encode ⁴	PUSHFQ (new mnemonic): Pushes the 64-bit RFLAGS register.
9C				
RCL —Rotate Through Carry Left	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /2				
D3 /2				
C1 /2				
RCR —Rotate Through Carry Right	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /3				
D3 /3				
C1 /3				
RDMSR —Read Model-Specific Register	Same as legacy mode.	Not relevant.	RDX[31:00] contains MSR[63:32], RAX[31:00] contains MSR[31:00]. Zero-extends 32-bit register results to 64 bits.	
0F 32				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, "General Rules for 64-Bit Mode", on page 1163 for definitions of "Promoted to 64 bits" and related topics.
2. If "Type of Operation" is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, "General Rules for 64-Bit Mode", on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
RDPMC —Read Performance-Monitoring Counters	Same as legacy mode.	Not relevant.	RDX[31:00] contains MSR[63:32], RAX[31:00] contains MSR[31:00]. Zero- extends 32-bit register results to 64 bits.	
0F 33				
RDTSC —Read Time-Stamp Counter	Same as legacy mode.	Not relevant.	RDX[31:00] contains MSR[63:32], RAX[31:00] contains MSR[31:00]. Zero- extends 32-bit register results to 64 bits.	
0F 31				
REP INS —Repeat Input String	Same as legacy mode.	32 bits	See footnote ⁵	
F3 6D				
REP LODS —Repeat Load String	Promoted to 64 bits.	32 bits	Zero-extends EAX, to 64 bits. See footnote ⁵	See footnote ⁵
F3 AD				
REP MOVS —Repeat Move String	Promoted to 64 bits.	32 bits	See footnote ⁵	
F3 A5				
REP OUTS —Repeat Output String to Port	Same as legacy mode.	32 bits	See footnote ⁵	
F3 6F				
REP STOS —Repeat Store String	Promoted to 64 bits.	32 bits	See footnote ⁵	
F3 AB				
REP_x CMPS —Repeat Compare String	Promoted to 64 bits.	32 bits	See footnote ⁵	
F3 A7				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
REPx SCAS –Repeat Scan String	Promoted to 64 bits.	32 bits	Zero-extends EAX, to 64 bits. See footnote ⁵	See footnote ⁵
F3 AF				
RET –Return from Call Near	See “Near Branches in 64-Bit Mode” on page 105 of Volume 1.			
C2	Promoted to 64 bits.	64 bits	No GPR results.	
C3				
RET –Return from Call Far	Promoted to 64 bits.	32 bits	See Section 3.7, “Control Transfers”, on page 95 of Volume 1 and Section 4.10, “Control-Transfer Privilege Checks”, on page 133 of Volume 2.	
CB				
CA				
ROL –Rotate Left	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /0				
D3 /0				
C1 /0				
ROR –Rotate Right	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /1				
D3 /1				
C1 /1				
Note: <div><div>1.</div><div>The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2.</div><div>If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3.</div><div>Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4.</div><div>The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5.</div><div>Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6.</div><div>See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
RSM —Resume from System Management Mode	New SMM state-save area.	Not relevant.	See Section 10, “System-Management Mode”, on page 331 of Volume 2.	
0F AA				
SAHF - Store AH into Flags	INVALID IN 64-BIT MODE (invalid-opcode exception)			
9E				
SAL —Shift Arithmetic Left	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /4				
D3 /4				
C1 /4				
SAR —Shift Arithmetic Right	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /7				
D3 /7				
C1 /7				
SBB —Subtract with Borrow	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
19				
1B				
1D				
81 /3				
83 /3				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
SCAS, SCASB, SCASW, SCASD, SCASQ —Scan String	Promoted to 64 bits.	32 bits	SCASD: Scan String Doublewords. Zero-extends 32-bit register results to 64 bits. See footnote ⁵	SCASQ (new mnemonic): Scan String Quadwords. See footnote ⁵
AF				
SETcc —Byte Set if Condition	Same as legacy mode.	Operand size fixed at 8 bits.		
0F 90 through 0F 9F				
SFENCE —Store Fence	Same as legacy mode.	Not relevant.	No GPR register results.	
0F AE /7				
SGDT —Store Global Descriptor Table Register	Promoted to 64 bits.	Operand size fixed at 8+2 bytes.	No GPR register results. Stores 8-byte base and 2-byte limit.	
0F 01 /0				
SHL —Shift Left	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /4				
D3 /4				
C1 /4				
SHLD —Shift Left Double	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
0F A4				
0F A5				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
SHR —Shift Right	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
D1 /5				
D3 /5				
C1 /5				
SHRD —Shift Right Double	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	Uses 6-bit count.
0F AC				
0F AD				
SIDT —Store Interrupt Descriptor Table Register	Promoted to 64 bits.	Operand size fixed at 8+2 bytes.	No GPR register results. Stores 8-byte base and 2-byte limit.	
0F 01 /1				
SLDT —Store Local Descriptor Table Register	Same as legacy mode.	32	Zero-extends 2-byte LDT selector to 64 bits.	
0F 00 /0				
SMSW —Store Machine Status Word	Same as legacy mode.	32	Zero-extends 2-byte MSW to 64 bits.	
0F 01 /4				
STC —Set Carry Flag	Same as legacy mode.	Not relevant.	No GPR register results.	
F9				
STD —Set Direction Flag	Same as legacy mode.	Not relevant.	No GPR register results.	
FD				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
STI - Set Interrupt Flag	Same as legacy mode.	Not relevant.	No GPR register results.	
FB				
STOS, STOSB, STOSW, STOSD, STOSQ -Store String	Promoted to 64 bits.	32 bits	STOSD: Store String Doublewords. See footnote ⁵	STOSQ (new mnemonic): Store String Quadwords. See footnote ⁵
AB				
STR —Store Task Register	Same as legacy mode.	32	Zero-extends 2-byte TR selector to 64 bits.	
0F 00 /1				
SUB —Subtract	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
29				
2B				
2D				
81 /5				
83 /5				
Note: <div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div> <div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div> <div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div> <div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div> <div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div> <div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div>				

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
SWAPGS —Swap GS Register with KernelGSbase MSR	New instruction, available only in 64-bit mode. (In other modes, this opcode is invalid.)	Not relevant	See Section 6.1.2, “SWAPGS Instruction”, on page 195 of Volume 2.	
0F 01 /7				
SYSCALL —Fast System Call	Promoted to 64 bits.	Not relevant.	See Section , “SYSCALL and SYSRET Instructions”, on page 192 of Volume 2 for details.	
0F 05				
SYSENTER —System Call	INVALID IN LONG MODE (invalid-opcode exception)			
0F 34				
SYSEXIT —System Return	INVALID IN LONG MODE (invalid-opcode exception)			
0F 35				
SYSRET —Fast System Return	Promoted to 64 bits.	32 bits	See Section , “SYSCALL and SYSRET Instructions”, on page 192 of Volume 2 for details.	
0F 07				
TEST —Test Bits	Promoted to 64 bits.	32 bits	No GPR register results.	
85				
A9				
F7 /0				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
UD2 —Undefined Operation	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 0B				
VERR —Verify Segment for Reads	Same as legacy mode.	Operand size fixed at 16 bits	No GPR register results.	
0F 00 /4				
VERW —Verify Segment for Writes	Same as legacy mode.	Operand size fixed at 16 bits	No GPR register results.	
0F 00 /5				
WAIT —Wait for Interrupt	Same as legacy mode.	Not relevant.	No GPR register results.	
9B				
WBINVD —Writeback and Invalidate All Caches	Same as legacy mode.	Not relevant.	No GPR register results.	
0F 09				
WRMSR —Write to Model-Specific Register	Same as legacy mode.	Not relevant.	No GPR register results. MSR[63:32] = RDX[31:00] MSR[31:00] = RAX[31:00]	
0F 30				
XADD —Exchange and Add	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
0F C1				
XCHG —Exchange Register/Memory with Register	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
87				
90				

Note:

1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.
2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.
3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. immediates and branch displacements are sign-extended to 64 bits.
4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.
5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.
6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.

Table A-1. Operations and Operands in 64-Bit Mode (continued)

Instruction and Opcode (hex) ⁶	Type of Operation ¹	Default Operand Size ²	For 32-Bit Operand Size ³	For 64-Bit Operand Size ³
XLAT, XLATB - Table Look-up Translation	Same as legacy mode.	Operand size fixed at 8 bits.	Writes AL, preserves RAX[63:08].	
D7				
XOR —Logical Exclusive OR	Promoted to 64 bits.	32 bits	Zero-extends 32-bit register results to 64 bits.	
31				
33				
35				
81 /6				
83 /6				
Note: <div><div>1. The type of operation, excluding considerations of operand size or extension of results. See Section A.1, “General Rules for 64-Bit Mode”, on page 1163 for definitions of “Promoted to 64 bits” and related topics.</div><div>2. If “Type of Operation” is 64 bits, a REX prefix is needed for 64-bit operand size, unless the instruction size defaults to 64 bits. If the operand size is fixed, operand-size overrides are silently ignored.</div><div>3. Special actions in 64-bit mode, in addition to legacy-mode actions. Zero or sign extensions apply only to result operands, not source operands. Unless otherwise stated, 8-bit and 16-bit results leave the high 56 or 48 bits, respectively, of 64-bit destination registers unchanged. Immediates and branch displacements are sign-extended to 64 bits.</div><div>4. The default operand size can be overridden to 16 bits with 66h prefix, but there is no 32-bit operand-size override in 64-bit mode.</div><div>5. Any pointer registers (rDI, rSI) or count registers (rCX) are address-sized and default to 64 bits. For 32-bit address size, any pointer and count registers are zero-extended to 64 bits.</div><div>6. See Appendix A.1, “General Rules for 64-Bit Mode”, on page 1163, for opcodes that do not appear in this table.</div></div>				

A.3 Invalid and Reassigned Instructions in 64-Bit Mode

Table A-2 lists instructions that are illegal in 64-bit mode. Attempted use of these instructions generates an invalid-opcode exception (#UD).

Table A-3 lists instructions that are reassigned to different functions in 64-bit mode. Attempted use of these instructions generates the reassigned function.

Table A-4 lists instructions that are illegal in long mode. Attempted use of these instructions generates an invalid-opcode exception (#UD).

Table A-2. Invalid Instructions in 64-Bit Mode

Mnemonic	Opcode (hex)	Description
AAA	37	ASCII Adjust After Addition
AAD	D5	ASCII Adjust Before Division
AAM	D4	ASCII Adjust After Multiply
AAS	3F	ASCII Adjust After Subtraction
BOUND	62	Check Array Bounds
CALL (far)	9A	Procedure Call Far (far absolute)
DAA	27	Decimal Adjust after Addition
DAS	2F	Decimal Adjust after Subtraction
INTO	CE	Interrupt to Overflow Vector
JMP (far)	EA	Jump Far (absolute)
LAHF	9F	Load Status Flags into AH Register
LDS	C5	Load DS Far Pointer
LES	C4	Load ES Far Pointer
POP DS	1F	Pop Stack into DS Segment
POP ES	07	Pop Stack into ES Segment
POP SS	17	Pop Stack into SS Segment
POPA, POPAD	61	Pop All to GPR Words or Doublewords
PUSH CS	0E	Push CS Segment Selector onto Stack
PUSH DS	1E	Push DS Segment Selector onto Stack
PUSH ES	06	Push ES Segment Selector onto Stack
PUSH SS	16	Push SS Segment Selector onto Stack
PUSHA, PUSHAD	60	Push All to GPR Words or Doublewords
SAHF	9E	Store AH into Flags
Redundant Grp1 (undocumented)	82	Redundant encoding of group1 Eb,lb opcodes
SALC (undocumented)	D6	Set AL According to CF

Table A-3. Reassigned Instructions in 64-Bit Mode

Mnemonic	Opcode (hex)	Description
ARPL	63	Opcode for MOVSLD instruction in 64-bit mode. In all other modes, this the Adjust Requestor Privilege Level instruction opcode.
DEC and INC	40-4F	Decrement by 1, Increment by 1 (see “Single-Byte INC and DEC Instructions in 64-Bit Mode” on page 1199). Two-byte versions of DEC and INC are still valid.

Table A-4. Invalid Instructions in Long Mode

Mnemonic	Opcode (hex)	Description
SYSENTER	0F 34	System Call
SYSEXIT	0F 35	System Return

A.4 Instructions with 64-Bit Default Operand Size

In 64-bit mode, two groups of instructions default to 64-bit operand size without the need for a REX prefix:

- *Near branches*—CALL, Jcc, JMP, LOOP, and RET.
- *All instructions, except far branches, that implicitly reference the RSP*—CALL, ENTER, LEAVE, POP, PUSH, and RET (CALL and RET are in both groups of instructions).

Table A-5 lists these instructions.

Table A-5. Instructions Defaulting to 64-Bit Operand Size

Mnemonic	Opcode (hex)	Implicitly Reference RSP	Description
CALL	E8, FF /2	yes	Call Procedure Near
ENTER	C8	yes	Create Procedure Stack Frame
Jcc	many	no	Jump Conditional Near
JMP	E9, EB, FF /4	no	Jump Near
LEAVE	C9	yes	Delete Procedure Stack Frame
LOOP	E2	no	Loop
LOOPcc	E0, E1	no	Loop Conditional
POP reg/mem	8F /0	yes	Pop Stack (register or memory)
POP reg	58-5F	yes	Pop Stack (register)
POP FS	0F A1	yes	Pop Stack into FS Segment Register
POP GS	0F A9	yes	Pop Stack into GS Segment Register
POPF, POPFD, POPFQ	9D	yes	Pop to rFLAGS Word, Doubleword, or Quadword
PUSH imm8	6A	yes	Push onto Stack (sign-extended byte)
PUSH imm32	68	yes	Push onto Stack (sign-extended doubleword)
PUSH reg/mem	FF /6	yes	Push onto Stack (register or memory)
PUSH reg	50-57	yes	Push onto Stack (register)
PUSH FS	0F A0	yes	Push FS Segment Register onto Stack

Table A-5. Instructions Defaulting to 64-Bit Operand Size (continued)

Mnemonic	Opcode (hex)	Implicitly Reference RSP	Description
PUSH GS	0F A8	yes	Push GS Segment Register onto Stack
PUSHF, PUSHFD, PUSHFQ	9C	yes	Push rFLAGS Word, Doubleword, or Quadword onto Stack
RET	C2, C3	yes	Return From Call (near)

The 64-bit default operand size can be overridden to 16 bits using the 66h operand-size override. However, it is not possible to override the operand size to 32 bits because there is no 32-bit operand-size override prefix for 64-bit mode. See “Operand-Size Override Prefix” on page 5 for details.

A.5 Single-Byte INC and DEC Instructions in 64-Bit Mode

In 64-bit mode, the legacy encodings for the 16 single-byte INC and DEC instructions (one for each of the eight GPRs) are used to encode the REX prefix values, as described in “REX Prefixes” on page 14. Therefore, these single-byte opcodes for INC and DEC are not available in 64-bit mode, although they are available in legacy and compatibility modes. The functionality of these INC and DEC instructions is still available in 64-bit mode, however, using the ModRM forms of those instructions (opcodes FF/0 and FF/1).

A.6 NOP in 64-Bit Mode

Programs written for the legacy x86 architecture commonly use opcode 90h (the XCHG EAX, EAX instruction) as a one-byte NOP. In 64-bit mode, the processor treats opcode 90h specially in order to preserve this legacy NOP use. Without special handling in 64-bit mode, the instruction would not be a true no-operation. Therefore, in 64-bit mode the processor treats XCHG EAX, EAX as a true NOP, regardless of operand size or the presence of a REX prefix.

This special handling does not apply to the two-byte ModRM form of the XCHG instruction. Unless a 64-bit operand size is

specified using a REX prefix byte, using the two byte form of XCHG to exchange a register with itself will not result in a no-operation, because the default operation size is 32 bits in 64-bit mode.

A.7 Segment Override Prefixes in 64-Bit Mode

In 64-bit mode, the CS, DS, ES, SS segment-override prefixes have no effect. These four prefixes are no longer treated as segment-override prefixes in the context of multiple-prefix rules. Instead, they are treated as null prefixes.

The FS and GS segment-override prefixes are treated as true segment-override prefixes in 64-bit mode. Use of the FS and GS prefixes cause their respective segment bases to be added to the effective address calculation. See “FS and GS Registers in 64-Bit Mode” on page 97 of Volume 2 for details.

Appendix B Differences Between Long Mode and Legacy Mode

Table B-1 summarizes the major differences between 64-bit mode and legacy protected mode. The third column indicates differences between 64-bit mode and legacy mode. The fourth column indicates whether that difference also applies to compatibility mode.

Table B-1. Differences Between Long Mode and Legacy Mode

Type	Subject	64-Bit Mode Difference	Applies To Compatibility Mode?
Application Programming	Addressing	RIP-relative addressing available	no
	Data and Address Sizes	Default data size is 32 bits	
		REX Prefix toggles data size to 64 bits	
		Default address size is 64 bits	
		Address size prefix toggles address size to 32 bits	
	Instruction Differences	Various opcodes are invalid or changed (see Table A-4 on page 1197)	
		MOV reg,imm32 becomes MOV reg,imm64 (with REX operand size prefix)	
		REX is always enabled	
		Direct-offset forms of MOV to or from accumulator become 64-bit offsets	
		MOVD extended to MOV 64 bits between MMX registers and long GPRs (with REX operand-size prefix)	

Table B-1. Differences Between Long Mode and Legacy Mode (continued)

Type	Subject	64-Bit Mode Difference	Applies To Compatibility Mode?
System Programming	x86 Modes	Real and virtual-8086 modes not supported	yes
	Task Switching	Task switching not supported	yes
	Addressing	64-bit virtual addresses	yes
		4-level paging structures	
		PAE must always be enabled	
	Segmentation	CS, DS, ES, SS segment bases are ignored	no
		CS, DS, ES, FS, GS, SS segment limits are ignored	
		CS, DS, ES, SS Segment prefixes are ignored	
	Exception and Interrupt Handling	All pushes are 8 bytes	yes
		IDT entries are expanded to 16 bytes	
		SS is not changed for stack switch	
		SS:RSP is pushed unconditionally	
	Call Gates	All pushes are 8 bytes	yes
		16-bit call gates are illegal	
		32-bit call gate type is redefined as 64-bit call gate and is expanded to 16 bytes.	
		SS is not changed for stack switch	
	System-Descriptor Registers	GDT, IDT, LDT, TR base registers expanded to 64 bits	yes
	System-Descriptor Table Entries and Pseudo-descriptors	LGDT and LIDT use expanded 10-byte pseudo-descriptors.	no
		LLDT and LTR use expanded 16-byte table entries.	

Appendix C Instruction Subsets and CUID Feature Sets

Table C-1 is an alphabetical list of the x86-64 instruction set, including the instructions from all five of the instruction subsets that make up the entire x86-64 instruction-set architecture:

- Section 2.6, “General-Purpose Instruction Reference”, on page 57.
- Section 2.7, “128-Bit Media Instruction Reference”, on page 291.
- Section 2.8, “64-Bit Media Instruction Reference”, on page 681.
- Section 2.9, “x87 Floating-Point Instruction Reference”, on page 909.
- Section 2.10, “System Instruction Reference”, on page 1047.

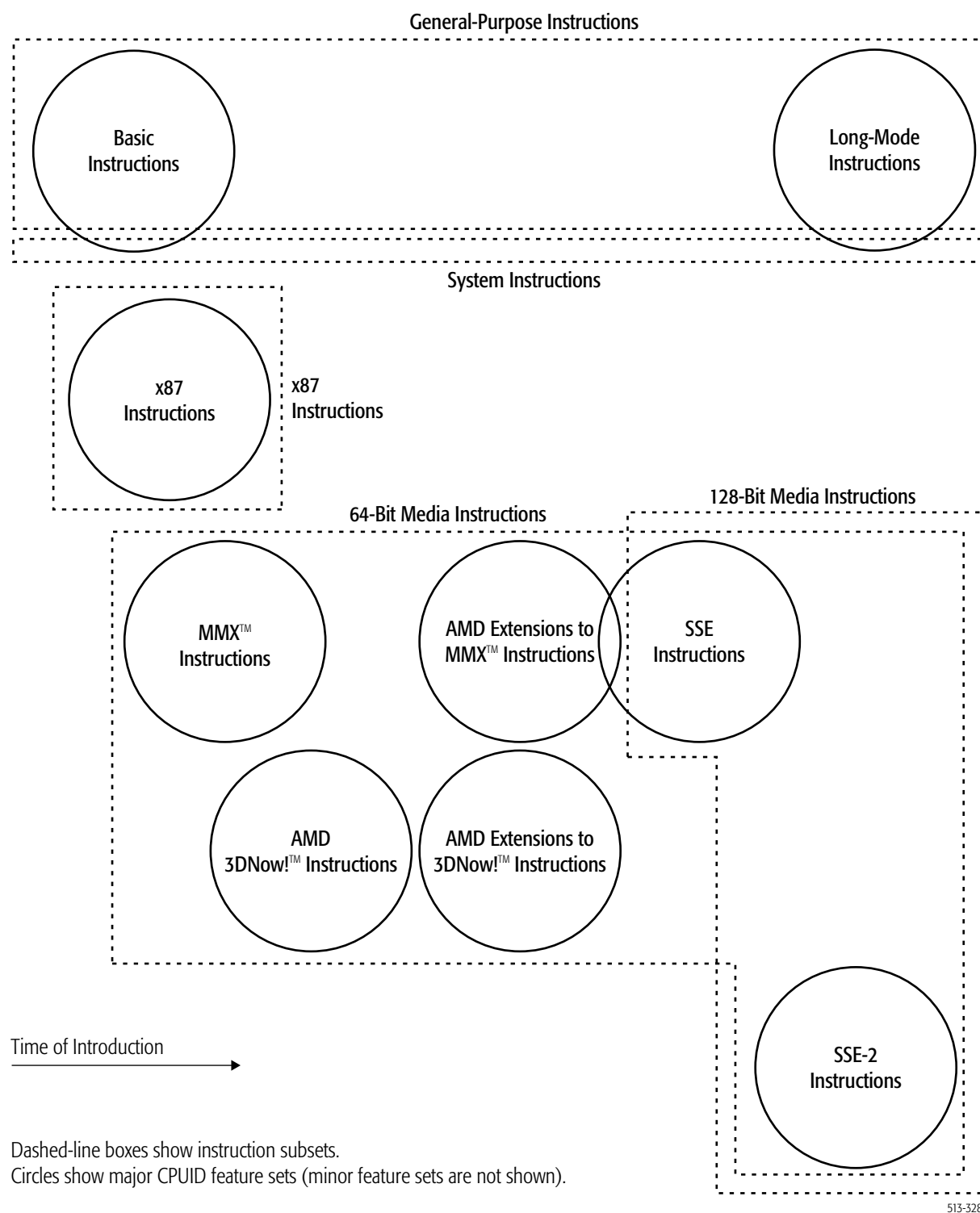
Several instructions belong to—and are described in—multiple instruction subsets. Table C-1 shows the minimum current privilege level (CPL) required to execute each instruction and the instruction subset(s) to which the instruction belongs. For each instruction subset, the CUID feature set(s) that enables the instruction is shown.

C.1 Instruction Subsets

Figure C-1 shows the relationship between the five instruction subsets and the CUID feature sets. Dashed-line polygons represent the instruction subsets. Circles represent the major CUID feature sets that enable various classes of instructions. (There are a few additional CUID feature sets, not shown, each of which apply to only a few instructions.)

The overlapping of the 128-bit and 64-bit media instruction subsets indicates that these subsets share some common mnemonics. However, these common mnemonics either have distinct opcodes for each subset or they take operands in both the MMX and XMM register sets.

The horizontal axis of Figure C-1 shows how the subsets and CUID feature sets have evolved over time.

**Figure C-1. Instruction Subsets vs. CPUID Feature Sets**

C.2 CPUID Feature Sets

The CPUID feature sets shown in Figure C-1 and listed in Table C-1 on page 1207 include:

- *Basic Instructions*—Instructions that are supported in all hardware implementations of the x86-64 architecture, except that the following instructions are implemented only if their associated CPUID function bit is set:
 - CMPXCHG8B, indicated by bit 8 of CPUID standard function 1 and extended function 8000_0001h.
 - CMOVcc (conditional moves), indicated by bit 15 of CPUID standard function 1 and extended function 8000_0001h.
 - RDMSR and WRMSR, indicated by bit 5 of CPUID standard function 1 and extended function 8000_0001h.
 - SYSCALL and SYSRET, indicated by bit 11 of CPUID extended function 8000_0001h.
 - SYSENTER and SYSEXIT, indicated by bit 11 of CPUID standard function 1.
- *x87 Instructions*—Legacy floating-point instructions that use the ST(0)–ST(7) stack registers (FPR0–FPR7 physical registers) and are supported if the following bits are set:
 - On-chip floating-point unit, indicated by bit 0 of CPUID standard function 1 and extended function 8000_0001h.
 - CMOVcc (conditional moves), indicated by bit 15 of CPUID standard function 1 and extended function 8000_0001h. This bit indicates support for x87 floating-point conditional moves (FCMOVcc) whenever the On-Chip Floating-Point Unit bit (bit 0) is also set.
- *MMX™ Instructions*—Vector integer instructions that are implemented in the MMX instruction set, use the MMX logical registers (FPR0–FPR7 physical registers), and are supported if the following bit is set:
 - MMX instructions, indicated by bit 23 of CPUID standard function 1 and extended function 8000_0001h.
- *AMD 3DNow!™ Instructions*—Vector floating-point instructions that comprise the AMD 3DNow! technology, use the MMX logical registers (FPR0–FPR7 physical registers), and are supported if the following bit is set:

- AMD 3DNow! instructions, indicated by bit 31 of CUID extended function 8000_0001h.
- *AMD Extensions to MMX™ Instructions*—Vector integer instructions that use the MMX registers and are supported if the following bit is set:
 - AMD extensions to MMX instructions, indicated by bit 22 of CUID extended function 8000_0001h.
- *AMD Extensions to 3DNow!™ Instructions*—Vector floating-point instructions that use the MMX registers and are supported if the following bit is set:
 - AMD extensions to 3DNow! instructions, indicated by bit 30 of CUID extended function 8000_0001h.
- *SSE Instructions*—Vector integer instructions that use the MMX registers, single-precision vector and scalar floating-point instructions that use the XMM registers, plus other instructions for data-type conversion, prefetching, cache control, and memory-access ordering. These instructions are supported if the following bits are set:
 - SSE, indicated by bit 25 of CUID extended function 8000_0001h.
 - FXSAVE and FXRSTOR, indicated by bit 24 of CUID standard function 1 and extended function 8000_0001h.

Several SSE opcodes are also implemented by the AMD Extensions to MMX™ Instructions.

- *SSE-2 Instructions*—Vector and scalar integer and double-precision floating-point instructions that use the XMM registers, plus other instructions for data-type conversion, cache control, and memory-access ordering. These instructions are supported if the following bit is set:
 - SSE-2, indicated by bit 26 of CUID extended function 8000_0001h.

Several instructions originally implemented as MMX™ instructions are extended in the SSE-2 instruction set to include opcodes that use XMM registers.

- *Long-Mode Instructions*—Instructions introduced by AMD with the x86-64 architecture. These instructions are supported if the following bit is set:
 - Long mode, indicated by bit 29 of CUID extended function 8000_0001h.

For complete details on the CUID feature sets listed in Table C-1, see Section 3.3, “Processor Feature Identification”, on page 77 of Volume 2.

C.3 Instruction List

Table C-1. Instruction Subsets and CUID Feature Sets

Instruction			Instruction Subset and CUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
AAA	ASCII Adjust After Addition	3	Basic				
AAD	ASCII Adjust Before Division	3	Basic				
AAM	ASCII Adjust After Multiply	3	Basic				
AAS	ASCII Adjust After Subtraction	3	Basic				
ADC	Add with Carry	3	Basic				
ADD	Signed or Unsigned Add	3	Basic				
ADDPD	Add Packed Double-Precision Floating-Point	3		SSE-2			
ADDPS	Add Packed Single-Precision Floating-Point	3		SSE			
ADDSD	Add Scalar Double-Precision Floating-Point	3		SSE-2			
ADDSS	Add Scalar Single-Precision Floating-Point	3		SSE			
AND	Logical AND	3	Basic				
ANDNPD	Logical Bitwise AND NOT Packed Double-Precision Floating-Point	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
ANDNPS	Logical Bitwise AND NOT Packed Single-Precision Floating-Point	3		SSE			
ANDPD	Logical Bitwise AND Packed Double-Precision Floating-Point	3		SSE-2			
ANDPS	Logical Bitwise AND Packed Single-Precision Floating-Point	3		SSE			
ARPL	Adjust Requestor Privilege Level	3					Basic
BOUND	Check Array Bounds	3	Basic				
BSF	Bit Scan Forward	3	Basic				
BSR	Bit Scan Reverse	3	Basic				
BSWAP	Byte Swap	3	Basic				
BT	Bit Test	3	Basic				
BTC	Bit Test and Complement	3	Basic				
BTR	Bit Test and Reset	3	Basic				
BTS	Bit Test and Set	3	Basic				
CALL	Procedure Call	3	Basic				
CBW	Convert Byte to Word	3	Basic				
CDQ	Convert Doubleword to Quadword	3	Basic				
CDQE	Convert Doubleword to Quadword	3	Long Mode				
CLC	Clear Carry Flag	3	Basic				
CLD	Clear Direction Flag	3	Basic				

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CLFLUSH	Cache Line Invalidate	3	SSE-2				
CLI	Clear Interrupt Flag	3					Basic
CLTS	Clear Task-Switched Flag in CR0	0					Basic
CMC	Complement Carry Flag	3	Basic				
CMOVA	Conditional Move If Above	3	CMOVcc				
CMOVAE	Conditional Move If Above or Equal	3	CMOVcc				
CMOVB	Conditional Move If Below	3	CMOVcc				
CMOVBE	Conditional Move If Below or Equal	3	CMOVcc				
CMOVC	Conditional Move If Carry	3	CMOVcc				
CMOVE	Conditional Move If Equal	3	CMOVcc				
CMOVG	Conditional Move If Greater	3	CMOVcc				
CMOVGE	Conditional Move If Greater or Equal	3	CMOVcc				
CMOVL	Conditional Move If Less	3	CMOVcc				
CMOVLE	Conditional Move If Less or Equal	3	CMOVcc				
CMOVNA	Conditional Move If Not Above	3	CMOVcc				
CMOVNAE	Conditional Move If Not Above or Equal	3	CMOVcc				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CMOVNB	Conditional Move If Not Below	3	CMOVcc				
CMOVNBE	Conditional Move If Not Below or Equal	3	CMOVcc				
CMOVNC	Conditional Move If Not Carry	3	CMOVcc				
CMOVNE	Conditional Move If Not Equal	3	CMOVcc				
CMOVNG	Conditional Move If Not Greater	3	CMOVcc				
CMOVNGE	Conditional Move If Not Greater or Equal	3	CMOVcc				
CMOVNL	Conditional Move If Not Less	3	CMOVcc				
CMOVNLE	Conditional Move If Not Less or Equal	3	CMOVcc				
CMOVNO	Conditional Move If No Overflow	3	CMOVcc				
CMOVNP	Conditional Move If No Parity	3	CMOVcc				
CMOVNS	Conditional Move If Not Sign	3	CMOVcc				
CMOVNZ	Conditional Move If Not Zero	3	CMOVcc				
CMOVO	Conditional Move If Overflow	3	CMOVcc				
CMOVP	Conditional Move If Parity	3	CMOVcc				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CMOVPE	Conditional Move If Parity Even	3	CMOVcc				
CMOVPO	Conditional Move If Parity Odd	3	CMOVcc				
CMOVS	Conditional Move If Sign	3	CMOVcc				
CMOVZ	Conditional Move If Zero	3	CMOVcc				
CMP	Compare	3	Basic				
CMPPD	Compare Packed Double-Precision Floating-Point	3		SSE-2			
CMPPS	Compare Packed Single-Precision Floating-Point	3		SSE			
CMPS	Compare Strings	3	Basic				
CMPSB	Compare Strings by Byte	3	Basic				
CMPSD	Compare Strings by Doubleword	3	Basic ²				
CMPSD	Compare Scalar Double-Precision Floating-Point	3		SSE-2 ²			
CMPSQ	Compare Strings by Quadword	3	Long Mode				
CMPSD	Compare Scalar Single-Precision Floating-Point	3		SSE			
CMPSW	Compare Strings by Word	3	Basic				
CMPXCHG	Compare and Exchange	3	Basic				
CMPXCHG8B	Compare and Exchange Eight Bytes	3	CMPXCHG8B				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
COMISD	Compare Ordered Scalar Double-Precision Floating-Point	3		SSE-2			
COMISS	Compare Ordered Scalar Single-Precision Floating-Point	3		SSE			
CPUID	Processor Identification	3	Basic				
CQO	Convert Quadword to Double Quadword	3	Long Mode				
CVTDQ2PD	Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point	3		SSE-2			
CVTDQ2PS	Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point	3		SSE-2			
CVTPD2DQ	Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers	3		SSE-2			
CVTPD2PI	Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers	3		SSE-2	SSE-2		
CVTPD2PS	Convert Packed Double-Precision Floating-Point to Packed Single-Precision Floating-Point	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CVTPI2PD	Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point	3		SSE-2	SSE-2		
CVTPI2PS	Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point	3		SSE	SSE		
CVTPS2DQ	Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers	3		SSE-2			
CVTPS2PD	Convert Packed Single-Precision Floating-Point to Packed Double-Precision Floating-Point	3		SSE-2			
CVTPS2PI	Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers	3		SSE	SSE		
CVTSD2SI	Convert Scalar Double-Precision Floating-Point to Signed Doubleword or Quadword Integer	3		SSE-2			
CVTSD2SS	Convert Scalar Double-Precision Floating-Point to Scalar Single-Precision Floating-Point	3		SSE-2			
CVTSI2SD	Convert Signed Doubleword or Quadword Integer to Scalar Double-Precision Floating-Point	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CVTSI2SS	Convert Signed Doubleword or Quadword Integer to Scalar Single-Precision Floating-Point	3		SSE			
CVTSS2SD	Convert Scalar Single-Precision Floating-Point to Scalar Double-Precision Floating-Point	3		SSE-2			
CVTSS2SI	Convert Scalar Single-Precision Floating-Point to Signed Doubleword or Quadword Integer	3		SSE			
CVTTPD2DQ	Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated	3		SSE-2			
CVTTPD2PI	Convert Packed Double-Precision Floating-Point to Packed Doubleword Integers, Truncated	3		SSE-2	SSE-2		
CVTTPS2DQ	Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated	3		SSE-2			
CVTTPS2PI	Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated	3		SSE	SSE		
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
CVTTSD2SI	Convert Scalar Double-Precision Floating-Point to Signed Doubleword or Quadword Integer, Truncated	3		SSE-2			
CVTTSS2SI	Convert Scalar Single-Precision Floating-Point to Signed Doubleword or Quadword Integer, Truncated	3		SSE			
CWD	Convert Word to Doubleword	3	Basic				
CWDE	Convert Word to Doubleword	3	Basic				
DAA	Decimal Adjust after Addition	3	Basic				
DAS	Decimal Adjust after Subtraction	3	Basic				
DEC	Decrement by 1	3	Basic				
DIV	Unsigned Divide	3	Basic				
DIVPD	Divide Packed Double-Precision Floating-Point	3		SSE-2			
DIVPS	Divide Packed Single-Precision Floating-Point	3		SSE			
DIVSD	Divide Scalar Double-Precision Floating-Point	3		SSE-2			
DIVSS	Divide Scalar Single-Precision Floating-Point	3		SSE			
EMMS	Enter/Exit Multimedia State	3			MMX	MMX	

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
ENTER	Create Procedure Stack Frame	3	Basic				
F2XM1	Floating-Point Compute $2x-1$	3				X87	
FABS	Floating-Point Absolute Value	3				X87	
FADD	Floating-Point Add	3				X87	
FADDP	Floating-Point Add and Pop	3				X87	
FBLD	Floating-Point Load Binary-Coded Decimal	3				X87	
FBSTP	Floating-Point Store Binary-Coded Decimal Integer and Pop	3				X87	
FCHS	Floating-Point Change Sign	3				X87	
FCLEX	Floating-Point Clear Flags	3				X87	
FCMOVB	Floating-Point Conditional Move If Below	3				X87, CMOVcc	
FCMOVBE	Floating-Point Conditional Move If Below or Equal	3				X87, CMOVcc	
FCMOVE	Floating-Point Conditional Move If Equal	3				X87, CMOVcc	
FCMOVNB	Floating-Point Conditional Move If Not Below	3				X87, CMOVcc	

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.
2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FCMOVNBE	Floating-Point Conditional Move If Not Below or Equal	3				X87, CMOVcc	
FCMOVNE	Floating-Point Conditional Move If Not Equal	3				X87, CMOVcc	
FCMOVNU	Floating-Point Conditional Move If Not Unordered	3				X87, CMOVcc	
FCMOVU	Floating-Point Conditional Move If Unordered	3				X87, CMOVcc	
FCOM	Floating-Point Compare	3				X87	
FCOMI	Floating-Point Compare and Set Flags	3				X87	
FCOMIP	Floating-Point Compare and Set Flags and Pop	3				X87	
FCOMP	Floating-Point Compare and Pop	3				X87	
FCOMPP	Floating-Point Compare and Pop Twice	3				X87	
FCOS	Floating-Point Cosine	3				X87	
FDECSTP	Floating-Point Decrement Stack-Top Pointer	3				X87	
FDIV	Floating-Point Divide	3				X87	
FDIVP	Floating-Point Divide and Pop	3				X87	
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FDIVR	Floating-Point Divide Reverse	3				X87	
FDIVRP	Floating-Point Divide Reverse and Pop	3				X87	
FEMMS	Fast Enter/Exit Multimedia State	3			3DNow!	3DNow!	
FFREE	Free Floating-Point Register	3				X87	
FIADD	Floating-Point Add Integer to Stack Top	3				X87	
FICOM	Floating-Point Integer Compare	3				X87	
FICOMP	Floating-Point Integer Compare and Pop	3				X87	
FIDIV	Floating-Point Integer Divide	3				X87	
FIDIVR	Floating-Point Integer Divide Reverse	3				X87	
FILD	Floating-Point Load Integer	3				X87	
FIMUL	Floating-Point Integer Multiply	3				X87	
FINCSTP	Floating-Point Increment Stack-Top Pointer	3				X87	
FINIT	Floating-Point Initialize	3				X87	
FIST	Floating-Point Integer Store	3				X87	
FISTP	Floating-Point Integer Store and Pop	3				X87	
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FISUB	Floating-Point Integer Subtract	3				X87	
FISUBR	Floating-Point Integer Subtract Reverse	3				X87	
FLD	Floating-Point Load	3				X87	
FLD1	Floating-Point Load +1.0	3				X87	
FLDCW	Floating-Point Load x87 Control Word	3				X87	
FLDENV	Floating-Point Load x87 Environment	3				X87	
FLDL2E	Floating-Point Load Log ₂ e	3				X87	
FLDL2T	Floating-Point Load Log ₂ 10	3				X87	
FLDLG2	Floating-Point Load Log ₁₀ 2	3				X87	
FLDLN2	Floating-Point Load Ln 2	3				X87	
FLDPI	Floating-Point Load Pi	3				X87	
FLDZ	Floating-Point Load +0.0	3				X87	
FMUL	Floating-Point Multiply	3				X87	
FMULP	Floating-Point Multiply and Pop	3				X87	
FNCLEX	Floating-Point No-Wait Clear Flags	3				X87	
FNINIT	Floating-Point No-Wait Initialize	3				X87	
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FNOP	Floating-Point No Operation	3				X87	
FNSAVE	Save No-Wait x87 and MMX State	3			X87	X87	
FNSTCW	Floating-Point No-Wait Store x87 Control Word	3				X87	
FNSTENV	Floating-Point No-Wait Store x87 Environment	3				X87	
FNSTSW	Floating-Point No-Wait Store x87 Status Word	3				X87	
FPATAN	Floating-Point Partial Arctangent	3				X87	
FPREM	Floating-Point Partial Remainder	3				X87	
FPREM1	Floating-Point Partial Remainder	3				X87	
FPTAN	Floating-Point Partial Tangent	3				X87	
FRNDINT	Floating-Point Round to Integer	3				X87	
FRSTOR	Restore x87 and MMX State	3			X87	X87	
FSAVE	Save x87 and MMX State	3			X87	X87	
FSCALE	Floating-Point Scale	3				X87	
FSIN	Floating-Point Sine	3				X87	
FSINCOS	Floating-Point Sine and Cosine	3				X87	
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FSQRT	Floating-Point Square Root	3				X87	
FST	Floating-Point Store Stack Top	3				X87	
FSTCW	Floating-Point Store x87 Control Word	3				X87	
FSTENV	Floating-Point Store x87 Environment	3				X87	
FSTP	Floating-Point Store Stack Top and Pop	3				X87	
FSTSW	Floating-Point Store x87 Status Word	3				X87	
FSUB	Floating-Point Subtract	3				X87	
FSUBP	Floating-Point Subtract and Pop	3				X87	
FSUBR	Floating-Point Subtract Reverse	3				X87	
FSUBRP	Floating-Point Subtract Reverse and Pop	3				X87	
FTST	Floating-Point Test with Zero	3				X87	
FUCOM	Floating-Point Unordered Compare	3				X87	
FUCOMI	Floating-Point Unordered Compare and Set Flags	3				X87	
FUCOMIP	Floating-Point Unordered Compare and Set Flags and Pop	3				X87	
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
FUCOMP	Floating-Point Unordered Compare and Pop	3				X87	
FUCOMPP	Floating-Point Unordered Compare and Pop Twice	3				X87	
FWAIT	Wait for x87 Floating-Point Exceptions	3				X87	
FXAM	Floating-Point Examine	3				X87	
FXCH	Floating-Point Exchange	3				X87	
FXRSTOR	Restore XMM, MMX, and x87 State	3		FXSAVE, FXRSTOR	FXSAVE, FXRSTOR	FXSAVE, FXRSTOR	
FXSAVE	Save XMM, MMX, and x87 State	3		FXSAVE, FXRSTOR	FXSAVE, FXRSTOR	FXSAVE, FXRSTOR	
FXTRACT	Floating-Point Extract Exponent and Significand	3				X87	
FYL2X	Floating-Point $y * \log_2 x$	3				X87	
FYL2XP1	Floating-Point $y * \log_2(x + 1)$	3				X87	
HLT	Halt	0					Basic
IDIV	Signed Divide	3	Basic				
IMUL	Signed Multiply	3	Basic				
IN	Input from Port	3	Basic				
INC	Increment by 1	3	Basic				
INS	Input String	3	Basic				
INSB	Input String Byte	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
INSD	Input String Doubleword	3	Basic				
INSW	Input String Word	3	Basic				
INT	Interrupt to Vector	3	Basic				
INT 3	Interrupt to Debug Vector	3					Basic
INTO	Interrupt to Overflow Vector	3	Basic				
INVD	Invalidate Caches	0					Basic
INVLPG	Invalidate TLB Entry	0					Basic
IRET	Interrupt Return Word	3					Basic
IRETD	Interrupt Return Doubleword	3					Basic
IRETQ	Interrupt Return Quadword	3					Long Mode
JA	Jump if Above	3	Basic				
JAЕ	Jump if Above or Equal	3	Basic				
JB	Jump if Below	3	Basic				
JBE	Jump if Below or Equal	3	Basic				
JC	Jump if Carry	3	Basic				
JCXZ	Jump if CX Zero	3	Basic				
JE	Jump if Equal	3	Basic				
JECXZ	Jump if ECX Zero	3	Basic				
JG	Jump if Greater	3	Basic				
JGE	Jump if Greater or Equal	3	Basic				
JL	Jump if Less	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
JLE	Jump if Less or Equal	3	Basic				
JMP	Jump	3	Basic				
JNA	Jump if Not Above	3	Basic				
JNAE	Jump if Not Above or Equal	3	Basic				
JNB	Jump if Not Below	3	Basic				
JNBE	Jump if Not Below or Equal	3	Basic				
JNC	Jump if Not Carry	3	Basic				
JNE	Jump if Not Equal	3	Basic				
JNG	Jump if Not Greater	3	Basic				
JNGE	Jump if Not Greater Or Equal	3	Basic				
JNL	Jump if Not Less	3	Basic				
JNLE	Jump if Not Less or Equal	3	Basic				
JNO	Jump if Not Overflow	3	Basic				
JNP	Jump if Not Parity	3	Basic				
JNS	Jump if Not Sign	3	Basic				
JNZ	Jump if Not Zero	3	Basic				
JO	Jump if Overflow	3	Basic				
JP	Jump if Parity	3	Basic				
JPE	Jump if Parity Even	3	Basic				
JPO	Jump if Parity Odd	3	Basic				
JRCXZ	Jump if RCX Zero	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
JS	Jump if Sign	3	Basic				
JZ	Jump if Zero	3	Basic				
LAHF	Load Status Flags into AH Register	3	Basic				
LAR	Load Access Rights Byte	3					Basic
LDMXCSR	Load MXCSR Control/Status Register	3		SSE			
LDS	Load DS Far Pointer	3	Basic				
LEA	Load Effective Address	3	Basic				
LEAVE	Delete Procedure Stack Frame	3	Basic				
LES	Load ES Far Pointer	3	Basic				
LFENCE	Load Fence	3	SSE-2				
LFS	Load FS Far Pointer	3	Basic				
LGDT	Load Global Descriptor Table Register	0					Basic
LGS	Load GS Far Pointer	3	Basic				
LIDT	Load Interrupt Descriptor Table Register	0					Basic
LLDT	Load Local Descriptor Table Register	0					Basic
LMSW	Load Machine Status Word	0					Basic
LODS	Load String	3	Basic				
LODSB	Load String Byte	3	Basic				
LODSD	Load String Doubleword	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
LODSQ	Load String Quadword	3	Long Mode				
LODSW	Load String Word	3	Basic				
LOOP	Loop	3	Basic				
LOOPE	Loop if Equal	3	Basic				
LOOPNE	Loop if Not Equal	3	Basic				
LOOPNZ	Loop if Not Zero	3	Basic				
LOOPZ	Loop if Zero	3	Basic				
LSL	Load Segment Limit	3	Basic				
LSS	Load SS Segment Register	3	Basic				
LTR	Load Task Register	0					Basic
MASKMOVDQU	Masked Move Double Quadword Unaligned	3		SSE-2			
MASKMOVQ	Masked Move Quadword	3			SSE, MMX Extensions		
MAXPD	Maximum Packed Double-Precision Floating-Point	3		SSE-2			
MAXPS	Maximum Packed Single-Precision Floating-Point	3		SSE			
MAXSD	Maximum Scalar Double-Precision Floating-Point	3		SSE-2			
MAXSS	Maximum Scalar Single-Precision Floating-Point	3		SSE			
MFENCE	Memory Fence	3	SSE-2				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
MINPD	Minimum Packed Double-Precision Floating-Point	3		SSE-2			
MINPS	Minimum Packed Single-Precision Floating-Point	3		SSE			
MINSD	Minimum Scalar Double-Precision Floating-Point	3		SSE-2			
MINSS	Minimum Scalar Single-Precision Floating-Point	3		SSE			
MOV	Move	3	Basic				
MOV (2)	Move to/from Control Registers	0					Basic
MOV (3)	Move to/from Debug Registers	0					Basic
MOVAPD	Move Aligned Packed Double-Precision Floating-Point	3		SSE-2			
MOVAPS	Move Aligned Packed Single-Precision Floating-Point	3		SSE			
MOVD	Move Doubleword or Quadword	3	MMX, SSE-2	SSE-2	MMX		
MOVDQ2Q	Move Quadword to Quadword	3		SSE-2	SSE-2		
MOVDQA	Move Aligned Double Quadword	3		SSE-2			
MOVDQU	Move Unaligned Double Quadword	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
MOVHLP	Move Packed Single-Precision Floating-Point High to Low	3		SSE			
MOVHPD	Move High Packed Double-Precision Floating-Point	3		SSE-2			
MOVHPS	Move High Packed Single-Precision Floating-Point	3		SSE			
MOVLHP	Move Packed Single-Precision Floating-Point Low to High	3		SSE			
MOVLPD	Move Low Packed Double-Precision Floating-Point	3		SSE-2			
MOVLPS	Move Low Packed Single-Precision Floating-Point	3		SSE			
MOVMSKPD	Extract Packed Double-Precision Floating-Point Sign Mask	3	SSE-2	SSE-2			
MOVMSKPS	Extract Packed Single-Precision Floating-Point Sign Mask	3	SSE	SSE			
MOVNTDQ	Move Non-Temporal Double Quadword	3		SSE-2			
MOVNTI	Move Non-Temporal Doubleword or Quadword	3	SSE-2				

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
MOVNTPD	Move Non-Temporal Packed Double-Precision Floating-Point	3		SSE-2			
MOVNTPS	Move Non-Temporal Packed Single-Precision Floating-Point	3		SSE			
MOVNTQ	Move Non-Temporal Quadword	3			SSE, MMX Extensions		
MOVQ	Move Quadword	3		SSE-2	MMX		
MOVQ2DQ	Move Quadword to Quadword	3		SSE-2	SSE-2		
MOVS	Move String	3	Basic				
MOVSB	Move String Byte	3	Basic				
MOVSD	Move String Doubleword	3	Basic ²				
MOVSD	Move Scalar Double-Precision Floating-Point	3		SSE-2 ²			
MOVSQ	Move String Quadword	3	Long Mode				
MOVSS	Move Scalar Single-Precision Floating-Point	3		SSE			
MOVSW	Move String Word	3	Basic				
MOVSBX	Move with Sign-Extend	3	Basic				
MOVSDX	Move with Sign-Extend Doubleword	3	Long Mode				
MOVUPD	Move Unaligned Packed Double-Precision Floating-Point	3		SSE-2			

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
MOVUPS	Move Unaligned Packed Single-Precision Floating-Point	3		SSE			
MOVZX	Move with Zero-Extend	3	Basic				
MUL	Multiply Unsigned	3	Basic				
MULPD	Multiply Packed Double-Precision Floating-Point	3		SSE-2			
MULPS	Multiply Packed Single-Precision Floating-Point	3		SSE			
MULSD	Multiply Scalar Double-Precision Floating-Point	3		SSE-2			
MULSS	Multiply Scalar Single-Precision Floating-Point	3		SSE			
NEG	Two's Complement Negation	3	Basic				
NOP	No Operation	3	Basic				
NOT	One's Complement Negation	3	Basic				
OR	Logical OR	3	Basic				
ORPD	Logical Bitwise OR Packed Double-Precision Floating-Point	3		SSE-2			
ORPS	Logical Bitwise OR Packed Single-Precision Floating-Point	3		SSE			
OUT	Output to Port	3	Basic				
OUTS	Output String	3	Basic				
OUTSB	Output String Byte	3	Basic				

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.
2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
OUTSD	Output String Doubleword	3	Basic				
OUTSW	Output String Word	3	Basic				
PACKSSDW	Pack with Saturation Signed Doubleword to Word	3		SSE-2	MMX		
PACKSSWB	Pack with Saturation Signed Word to Byte	3		SSE-2	MMX		
PACKUSWB	Pack with Saturation Signed Word to Unsigned Byte	3		SSE-2	MMX		
PADDB	Packed Add Bytes	3		SSE-2	MMX		
PADD	Packed Add Doublewords	3		SSE-2	MMX		
PADDQ	Packed Add Quadwords	3		SSE-2	SSE-2		
PADDSB	Packed Add Signed with Saturation Bytes	3		SSE-2	MMX		
PADDSW	Packed Add Signed with Saturation Words	3		SSE-2	MMX		
PADDUSB	Packed Add Unsigned with Saturation Bytes	3		SSE-2	MMX		
PADDUSW	Packed Add Unsigned with Saturation Words	3		SSE-2	MMX		
PADDW	Packed Add Words	3		SSE-2	MMX		
PAND	Packed Logical Bitwise AND	3		SSE-2	MMX		
PANDN	Packed Logical Bitwise AND NOT	3		SSE-2	MMX		
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PAVGB	Packed Average Unsigned Bytes	3		SSE-2	SSE, MMX Extensions		
PAVGUSB	Packed Average Unsigned Bytes	3			3DNow!		
PAVGW	Packed Average Unsigned Words	3		SSE-2	SSE, MMX Extensions		
PCMPEQB	Packed Compare Equal Bytes	3		SSE-2	MMX		
PCMPEQD	Packed Compare Equal Doublewords	3		SSE-2	MMX		
PCMPEQW	Packed Compare Equal Words	3		SSE-2	MMX		
PCMPGTB	Packed Compare Greater Than Signed Bytes	3		SSE-2	MMX		
PCMPGTD	Packed Compare Greater Than Signed Doublewords	3		SSE-2	MMX		
PCMPGTW	Packed Compare Greater Than Signed Words	3		SSE-2	MMX		
PEXTRW	Packed Extract Word	3		SSE-2	SSE, MMX Extensions		
PF2ID	Packed Floating-Point to Integer Doubleword Conversion	3			3DNow!		
PF2IW	Packed Floating-Point to Integer Word Conversion	3			3DNow! Extensions		
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PFACC	Packed Floating-Point Accumulate	3			3DNow!		
PFADD	Packed Floating-Point Add	3			3DNow!		
PFCMPEQ	Packed Floating-Point Compare Equal	3			3DNow!		
PFCMPGE	Packed Floating-Point Compare Greater or Equal	3			3DNow!		
PFCMPGT	Packed Floating-Point Compare Greater Than	3			3DNow!		
PFMAX	Packed Floating-Point Maximum	3			3DNow!		
PFMIN	Packed Floating-Point Minimum	3			3DNow!		
PFMUL	Packed Floating-Point Multiply	3			3DNow!		
PFNACC	Packed Floating-Point Negative Accumulate	3			3DNow! Extensions		
PFPNACC	Packed Floating-Point Positive-Negative Accumulate	3			3DNow! Extensions		
PFRCP	Packed Floating-Point Reciprocal Approximation	3			3DNow!		
PFRCPIT1	Packed Floating-Point Reciprocal, Iteration 1	3			3DNow!		
PFRCPIT2	Packed Floating-Point Reciprocal or Reciprocal Square Root, Iteration 2	3			3DNow!		
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PFRSQIT1	Packed Floating-Point Reciprocal Square Root, Iteration 1	3			3DNow!		
PFRSQRT	Packed Floating-Point Reciprocal Square Root Approximation	3			3DNow!		
PFSUB	Packed Floating-Point Subtract	3			3DNow!		
PFSUBR	Packed Floating-Point Subtract Reverse	3			3DNow!		
PI2FD	Packed Integer to Floating-Point Doubleword Conversion	3			3DNow!		
PI2FW	Packed Integer To Floating-Point Word Conversion	3			3DNow! Extensions		
PINSRW	Packed Insert Word	3		SSE-2	SSE, MMX Extensions		
PMADDWD	Packed Multiply Words and Add Doublewords	3		SSE-2	MMX		
PMAXSW	Packed Maximum Signed Words	3		SSE-2	SSE, MMX Extensions		
PMAXUB	Packed Maximum Unsigned Bytes	3		SSE-2	SSE, MMX Extensions		
PMINSW	Packed Minimum Signed Words	3		SSE-2	SSE, MMX Extensions		
PMINUB	Packed Minimum Unsigned Bytes	3		SSE-2	SSE, MMX Extensions		
PMOVMASKB	Packed Move Mask Byte	3		SSE-2	SSE, MMX Extensions		

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PMULHRW	Packed Multiply High Rounded Word	3			3DNow!		
PMULHUW	Packed Multiply High Unsigned Word	3		SSE-2	SSE, MMX Extensions		
PMULHW	Packed Multiply High Signed Word	3		SSE-2	MMX		
PMULLW	Packed Multiply Low Signed Word	3		SSE-2	MMX		
PMULUDQ	Packed Multiply Unsigned Doubleword and Store Quadword	3		SSE-2	SSE-2		
POP	Pop Stack	3	Basic				
POPA	Pop All to GPR Words	3	Basic				
POPAD	Pop All to GPR Doublewords	3	Basic				
POPF	Pop to FLAGS Word	3	Basic				
POPFD	Pop to EFLAGS Doubleword	3	Basic				
POPFQ	Pop to RFLAGS Quadword	3	Long Mode				
POR	Packed Logical Bitwise OR	3		SSE-2	MMX		
PREFETCH	Prefetch L1 Data-Cache Line	3	3DNow!				
PREFETCH/level	Prefetch Data to Cache Level /level/	3	SSE, MMX Extensions				
PREFETCHW	Prefetch L1 Data-Cache Line for Write	3	3DNow!				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PSADBW	Packed Sum of Absolute Differences of Bytes into a Word	3		SSE-2	SSE, MMX Extensions		
PSHUFD	Packed Shuffle Doublewords	3		SSE-2			
PSHUFHW	Packed Shuffle High Words	3		SSE-2			
PSHUFLW	Packed Shuffle Low Words	3		SSE-2			
PSHUFW	Packed Shuffle Words	3			SSE, MMX Extensions		
PSLLD	Packed Shift Left Logical Doublewords	3		SSE-2	MMX		
PSLLDQ	Packed Shift Left Logical Double Quadword	3		SSE-2			
PSLLQ	Packed Shift Left Logical Quadwords	3		SSE-2	MMX		
PSLLW	Packed Shift Left Logical Words	3		SSE-2	MMX		
PSRAD	Packed Shift Right Arithmetic Doublewords	3		SSE-2	MMX		
PSRAW	Packed Shift Right Arithmetic Words	3		SSE-2	MMX		
PSRLD	Packed Shift Right Logical Doublewords	3		SSE-2	MMX		
PSRLDQ	Packed Shift Right Logical Double Quadword	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PSRLQ	Packed Shift Right Logical Quadwords	3		SSE-2	MMX		
PSRLW	Packed Shift Right Logical Words	3		SSE-2	MMX		
PSUBB	Packed Subtract Bytes	3		SSE-2	MMX		
PSUBD	Packed Subtract Doublewords	3		SSE-2	MMX		
PSUBQ	Packed Subtract Quadword	3		SSE-2	SSE-2		
PSUBSB	Packed Subtract Signed With Saturation Bytes	3		SSE-2	MMX		
PSUBSW	Packed Subtract Signed with Saturation Words	3		SSE-2	MMX		
PSUBUSB	Packed Subtract Unsigned and Saturate Bytes	3		SSE-2	MMX		
PSUBUSW	Packed Subtract Unsigned and Saturate Words	3		SSE-2	MMX		
PSUBW	Packed Subtract Words	3		SSE-2	MMX		
PSWAPD	Packed Swap Doubleword	3			3DNow! Extensions		
PUNPCKHBW	Unpack and Interleave High Bytes	3		SSE-2	MMX		
PUNPCKHDQ	Unpack and Interleave High Doublewords	3		SSE-2	MMX		
PUNPCKHQDQ	Unpack and Interleave High Quadwords	3		SSE-2			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
PUNPCKHWD	Unpack and Interleave High Words	3		SSE-2	MMX		
PUNPCKLBW	Unpack and Interleave Low Bytes	3		SSE-2	MMX		
PUNPCKLDQ	Unpack and Interleave Low Doublewords	3		SSE-2	MMX		
PUNPCKLQDQ	Unpack and Interleave Low Quadwords	3		SSE-2			
PUNPCKLWD	Unpack and Interleave Low Words	3		SSE-2	MMX		
PUSH	Push onto Stack	3	Basic				
PUSHA	Push All GPR Words onto Stack	3	Basic				
PUSHAD	Push All GPR Doublewords onto Stack	3	Basic				
PUSHF	Push EFLAGS Word onto Stack	3	Basic				
PUSHFD	Push EFLAGS Doubleword onto Stack	3	Basic				
PUSHFQ	Push RFLAGS Quadword onto Stack	3	Long Mode				
PXOR	Packed Logical Bitwise Exclusive OR	3		SSE-2	MMX		
RCL	Rotate Through Carry Left	3	Basic				
RCPPS	Reciprocal Packed Single-Precision Floating-Point	3		SSE			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
RCPS	Reciprocal Scalar Single-Precision Floating-Point	3		SSE			
RCR	Rotate Through Carry Right	3	Basic				
RDMSR	Read Model-Specific Register	0					RDMSR, WRMSR
RDPMS	Read Performance-Monitoring Counter	3					Basic
RDTS	Read Time-Stamp Counter	3					Basic
RET	Return from Call	3	Basic				
ROL	Rotate Left	3	Basic				
ROR	Rotate Right	3	Basic				
RSM	Resume from System Management Mode	0					Basic
RSQRTS	Reciprocal Square Root Packed Single-Precision Floating-Point	3		SSE			
RSQRTSS	Reciprocal Square Root Scalar Single-Precision Floating-Point	3		SSE			
SAHF	Store AH into Flags	3	Basic				
SAL	Shift Arithmetic Left	3	Basic				
SAR	Shift Arithmetic Right	3	Basic				
SBB	Subtract with Borrow	3	Basic				
SCAS	Scan String	3	Basic				
SCASB	Scan String as Bytes	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
SCASD	Scan String as Doubleword	3	Basic				
SCASQ	Scan String as Quadword	3	Long Mode				
SCASW	Scan String as Words	3	Basic				
SETA	Set Byte if Above	3	Basic				
SETAE	Set Byte if Above or Equal	3	Basic				
SETB	Set Byte if Below	3	Basic				
SETBE	Set Byte if Below or Equal	3	Basic				
SETC	Set Byte if Carry	3	Basic				
SETE	Set Byte if Equal	3	Basic				
SETG	Set Byte if Greater	3	Basic				
SETGE	Set Byte if Greater or Equal	3	Basic				
SETL	Set Byte if Less	3	Basic				
SETLE	Set Byte if Less or Equal	3	Basic				
SETNA	Set Byte if Not Above	3	Basic				
SETNAE	Set Byte if Not Above or Equal	3	Basic				
SETNB	Set Byte if Not Below	3	Basic				
SETNBE	Set Byte if Not Below or Equal	3	Basic				
SETNC	Set Byte if No Carry	3	Basic				
SETNE	Set Byte if Not Equal	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
SETNG	Set Byte if Not Greater	3	Basic				
SETNGE	Set Byte if Not Greater or Equal	3	Basic				
SETNL	Set Byte if Not Less	3	Basic				
SETNLE	Set Byte if Not Less or Equal	3	Basic				
SETNO	Set Byte if Not Overflow	3	Basic				
SETNP	Set Byte if Not Parity	3	Basic				
SETNS	Set Byte if Not Sign	3	Basic				
SETNZ	Set Byte if Not Zero	3	Basic				
SETO	Set Byte if Overflow	3	Basic				
SETP	Set Byte if Parity	3	Basic				
SETPE	Set Byte if Parity Even	3	Basic				
SETPO	Set Byte if Parity Odd	3	Basic				
SETS	Set Byte if Sign	3	Basic				
SETZ	Set Byte if Zero	3	Basic				
SFENCE	Store Fence	3	SSE, MMX Extensions				
SGDT	Store Global Descriptor Table Register	3					Basic
SHL	Shift Left	3	Basic				
SHLD	Shift Left Double	3	Basic				
SHR	Shift Right	3	Basic				
SHRD	Shift Right Double	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
SHUFPD	Shuffle Packed Double-Precision Floating-Point	3		SSE-2			
SHUFPS	Shuffle Packed Single-Precision Floating-Point	3		SSE			
SIDT	Store Interrupt Descriptor Table Register	3					Basic
SLDT	Store Local Descriptor Table Register	3					Basic
SMSW	Store Machine Status Word	3					Basic
SQRTPD	Square Root Packed Double-Precision Floating-Point	3		SSE-2			
SQRTPS	Square Root Packed Single-Precision Floating-Point	3		SSE			
SQRTSD	Square Root Scalar Double-Precision Floating-Point	3		SSE-2			
SQRTSS	Square Root Scalar Single-Precision Floating-Point	3		SSE			
STC	Set Carry Flag	3	Basic				
STD	Set Direction Flag	3	Basic				
STI	Set Interrupt Flag	3					Basic
STMXCSR	Store MXCSR Control/Status Register	3		SSE			
STOS	Store String	3	Basic				
STOSB	Store String Bytes	3	Basic				

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.
2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
STOSD	Store String Doublewords	3	Basic				
STOSQ	Store String Quadwords	3	Long Mode				
STOSW	Store String Words	3	Basic				
STR	Store Task Register	3					Basic
SUB	Subtract	3	Basic				
SUBPD	Subtract Packed Double-Precision Floating-Point	3		SSE-2			
SUBPS	Subtract Packed Single-Precision Floating-Point	3		SSE			
SUBSD	Subtract Scalar Double-Precision Floating-Point	3		SSE-2			
SUBSS	Subtract Scalar Single-Precision Floating-Point	3		SSE			
SWAPGS	Swap GS Register with KernelGSbase MSR	0					Long Mode
SYSCALL	Fast System Call	3					SYSCALL, SYSRET
SYSENTER	System Call	3					SYSENTER, SYSEXIT
SYSEXIT	System Return	0					SYSENTER, SYSEXIT
SYSRET	Fast System Return	0					SYSCALL, SYSRET
TEST	Test Bits	3	Basic				
UCOMISD	Unordered Compare Scalar Double-Precision Floating-Point	3		SSE-2			

1. Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs.

2. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands.

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
UCOMISS	Unordered Compare Scalar Single-Precision Floating-Point	3		SSE			
UD2	Undefined Operation	3					Basic
UNPCKHPD	Unpack High Double-Precision Floating-Point	3		SSE-2			
UNPCKHPS	Unpack High Single-Precision Floating-Point	3		SSE			
UNPCKLPD	Unpack Low Double-Precision Floating-Point	3		SSE-2			
UNPCKLPS	Unpack Low Single-Precision Floating-Point	3		SSE			
VERR	Verify Segment for Reads	3					Basic
VERW	Verify Segment for Writes	3					Basic
WAIT	Wait for x87 Floating-Point Exceptions	3				X87	
WBINVD	Writeback and Invalidate Caches	0					Basic
WRMSR	Write to Model-Specific Register	0					RDMSR, WRMSR
XADD	Exchange and Add	3	Basic				
XCHG	Exchange	3	Basic				
XLAT	Translate Table Index	3	Basic				
XLATB	Translate Table Index (No Operands)	3	Basic				
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Table C-1. Instruction Subsets and CPUID Feature Sets (continued)

Instruction			Instruction Subset and CPUID Feature Set(s) ¹				
Mnemonic	Description	CPL	General-Purpose	128-Bit Media	64-Bit Media	x87	System
XOR	Exclusive OR	3	Basic				
XORPD	Logical Bitwise Exclusive OR Packed Double-Precision Floating-Point	3		SSE-2			
XORPS	Logical Bitwise Exclusive OR Packed Single-Precision Floating-Point	3		SSE			
<ol style="list-style-type: none"> Columns indicate the instruction subsets. Entries indicate the CPUID feature set(s) to which the instruction belongs. Mnemonic is used for two different instructions. Assemblers can distinguish them by the number and type of operands. 							

Appendix D Instruction Effects on rFLAGS

The flags in the rFLAGS register are described in Section 3.1.4, “Flags Register”, on page 37 of Volume 1 and Section 3.1.7, “RFLAGS Register”, on page 64 of Volume 2. Table D-1 summarizes the effect that instructions have on these flags. The table includes all instructions that affect the flags. Instructions not shown have no effect on rFLAGS.

The following codes are used within the table:

- 0—The flag is or can be cleared to 0.
- 1—The flag is or can be set to 1.
- Load—The flag is loaded with value from AH register.
- Mod—The flag is or can be modified.
- Pop—The flag is loaded with value popped off of the stack.
- Tst—The flag is tested.
- U—The effect on the flag is undefined.
- Gray shaded cells indicate that the flag is not affected by the instruction.

Table D-1. Instruction Effects on rFLAGS

Instruction Mnemonic	rFLAGS Mnemonic and Bit Number															
	ID 21	VIP 20	VIF 19	AC 18	VM 17	RF 16	NT 14	OF 11	DF 10	IF 9	TF 8	SF 7	ZF 6	AF 4	PF 2	CF 0
AAA AAS								U				U	U	Tst Mod	U	Mod
AAD AAM								U				Mod	Mod	U	Mod	U
ADC								Mod				Mod	Mod	Mod	Mod	Tst Mod
ADD								Mod				Mod	Mod	Mod	Mod	Mod
AND								0				Mod	Mod	U	Mod	0
ARPL													Mod			
BSF BSR								U				U	Mod	U	U	U

Table D-1. Instruction Effects on rFLAGS (continued)

Instruction Mnemonic	rFLAGS Mnemonic and Bit Number															
	ID 21	VIP 20	VIF 19	AC 18	VM 17	RF 16	NT 14	OF 11	DF 10	IF 9	TF 8	SF 7	ZF 6	AF 4	PF 2	CF 0
BT BTC BTR BTS								U				U	U	U	U	Mod
CLC																0
CLD									0							
CLI										0						
CMC																Mod
CMOV _{cc}								Tst				Tst	Tst		Tst	Tst
CMP								Mod				Mod	Mod	Mod	Mod	Mod
CMPS _x								Mod	Tst			Mod	Mod	Mod	Mod	Mod
CMPXCHG								Mod				Mod	Mod	Mod	Mod	Mod
CMPXCHG 8B														Mod		
COMISD COMISS								0				0	Mod	0	Mod	Mod
DAA DAS								U				Mod	Mod	Tst Mod	Mod	Tst Mod
DEC								Mod				Mod	Mod	Mod	Mod	
DIV								U				U	U	U	U	U
FCMOV _{cc}													Tst		Tst	Tst
FCOMI FCOMIP FUCOMI FUCOMIP													Mod		Mod	Mod
IDIV								U				U	U	U	U	U
IMUL								Mod				U	U	U	U	Mod
INC								Mod				Mod	Mod	Mod	Mod	
INS _x									Tst							

Table D-1. Instruction Effects on rFLAGS (continued)

Instruction Mnemonic	rFLAGS Mnemonic and Bit Number															
	ID 21	VIP 20	VIF 19	AC 18	VM 17	RF 16	NT 14	OF 11	DF 10	IF 9	TF 8	SF 7	ZF 6	AF 4	PF 2	CF 0
INT INT 3				0	0	0	0			0	0					
INTO							0	Tst			0					
IRET _x							Tst	Pop	Pop	Pop	Pop	Pop	Pop	Pop	Pop	Pop
Jcc								Tst				Tst	Tst		Tst	Tst
LAR													Mod			
LODS _x									Tst							
LOOPE LOOPNE													Tst			
LSL													Mod			
MOV(2) CR _n MOV(3) DR _n								U				U	U	U	U	U
MOVS _x									Tst							
MUL								Mod				U	U	U	U	Mod
NEG								Mod				Mod	Mod	Mod	Mod	Mod
OR								0				Mod	Mod	U	Mod	0
OUTS _x									Tst							
POPF _x							Pop	Pop	Pop	Pop	Pop	Pop	Pop	Pop	Pop	Pop
RCL 1								Mod								Tst Mod
RCL <i>count</i>								U								Tst Mod
RCR 1								Mod								Tst Mod
RCR <i>count</i>								U								Tst Mod
ROL 1								Mod								Mod

Table D-1. Instruction Effects on rFLAGS (continued)

Instruction Mnemonic	rFLAGS Mnemonic and Bit Number															
	ID 21	VIP 20	VIF 19	AC 18	VM 17	RF 16	NT 14	OF 11	DF 10	IF 9	TF 8	SF 7	ZF 6	AF 4	PF 2	CF 0
ROL <i>count</i>								U								Mod
ROR 1								Mod								Mod
ROR <i>count</i>								U								Mod
RSM						Mod	Mod	Mod	Mod	Mod	Mod	Mod	Mod	Mod	Mod	Mod
SAHF												Load	Load	Load	Load	Load
SAL 1								Mod				Mod	Mod	U	Mod	Mod
SAL <i>count</i>								U				Mod	Mod	U	Mod	Mod
SAR 1								Mod				Mod	Mod	U	Mod	Mod
SAR <i>count</i>								U				Mod	Mod	U	Mod	Mod
SBB								Mod				Mod	Mod	Mod	Mod	Tst Mod
SCASx								Mod	Tst			Mod	Mod	Mod	Mod	Mod
SETcc								Tst				Tst	Tst		Tst	Tst
SHLD SHRD								U				Mod	Mod	U	Mod	Mod
STC																1
STD									1							
STI										1						
STOSx									Tst							
SUB								Mod				Mod	Mod	Mod	Mod	Mod
SYSCALL					0					0						
SYSENTER					0	0				0						
SYSRET										1						
TEST								0				Mod	Mod	U	Mod	0
UCOMISD UCOMISS								0				0	Mod	0	Mod	Mod

Table D-1. Instruction Effects on rFLAGS (continued)

Instruction Mnemonic	rFLAGS Mnemonic and Bit Number															
	ID 21	VIP 20	VIF 19	AC 18	VM 17	RF 16	NT 14	OF 11	DF 10	IF 9	TF 8	SF 7	ZF 6	AF 4	PF 2	CF 0
VERR VERW													Mod			
XADD								Mod				Mod	Mod	Mod	Mod	Mod
XOR								0				Mod	Mod	U	Mod	0

Index

Numerics

16-bit mode.....	xxii
32-bit mode.....	xxii
64-bit mode.....	xxiii

A

AAA	58
AAD	59
AAM.....	61
AAS	62
ADC.....	63
ADD	66
ADDPD	293
ADDPS.....	295
address size prefix	6, 24
addressing	
byte registers.....	17
effective address....	1156, 1158, 1159, 1161
PC-relative	23
RIP-relative	xxviii, 23
ADDSD	297
ADDSS	300
AND	69
ANDNPD	303
ANDNPS	305
ANDPD	307
ANDPS.....	309
ARPL	1048

B

base field	1160, 1161
biased exponent.....	xxiii
BOUND	72
BSF	74
BSR	76
BSWAP	78
BT	79
BTC	81
BTR	83
BTS.....	85
byte order of instructions	1
byte register addressing.....	17

C

CALL.....	15, 87
CBW	94
CDQ.....	95
CDQE	96
CLC	97

CLD.....	98
CLFLUSH	99, 1141
CLI.....	1050
CLTS	1052
CMC.....	101
CMOVcc	102, 1137
CMP	106
CMPPD	311
CMPPS	314
CMPSD	317
CMPSS.....	320
CMPSx.....	109
CMPXCHG	112
CMPXCHG8B	15, 114
COMISD	323
COMISS.....	326
commit.....	xxiii
compatibility mode	xxiii
condition codes	
rFLAGS	1137, 1153
x87	909
count.....	1164
CPUID	116
extended functions	116
feature sets	1205
standard functions	116
CQO	117
CVTDQ2PD	329
CVTDQ2PS	331
CVTPD2DQ	333
CVTPD2PI.....	335, 684
CVTPD2PS.....	338
CVTPI2PD	341, 687
CVTPI2PS	343, 689
CVTPS2DQ	346
CVTPS2PD	348
CVTPS2PI	350, 692
CVTSD2SI	353
CVTSD2SS	356
CVTSI2SD	359
CVTSI2SS.....	362
CVTSS2SD	365
CVTSS2SI.....	367
CVTTPD2DQ	370
CVTTPD2PI.....	372, 695
CVTTPS2DQ	375
CVTTPS2PI.....	377, 698
CVTTSD2SI.....	380

CVTTSS2SI	383	FFREE	944
CWD	118	FICOM _x	945
CWDE	119	FILD	948
D		FINCSTP	950
DAA	120	FINIT _x	952
DAS	121	FIST _x	954
data types		FLD	957
128-bit media	32	FLD1	960
64-bit media	34	FLDCW	961
general-purpose	30	FLDENV	963
x87	36	FLDL2E	965
DEC	17, 122, 1199	FLDL2T	967
direct referencing	xxiv	FLDLG2	969
displacements	xxiv, 22, 1164	FLDLN2	971
DIV	124	FLDPI	973
DIVPD	386	FLDZ	975
DIVPS	389	flush	xxv
DIVSD	392	FMUL _x	976
DIVSS	395	FNOP	979
double quadword	xxiv	FNSAVE	703, 980
doubleword	xxiv	FPATAN	982
E		FPREM	984
eAX–eSP register	xxx	FPREM1	986
effective address	1156, 1158, 1159, 1161	FPTAN	988
effective address size	xxiv	FRNDINT	990
effective operand size	xxv	FRSTOR	705, 992
eFLAGS register	xxx	FSAVE	707, 994
eIP register	xxx	FSCALE	996
element	xxv	FSIN	998
EMMS	701	FSINCOS	1000
endian order	xxxii, 1	FSQRT	1002
ENTER	126	FST	1004
exceptions	xxv, 39	FSTCW _x	1007
exponent	xxiii	FSTENV _x	1009
F		FSTP	1011
F2XM1	910	FSTSW _x	1014
FABS	912	FSUBR _x	1019
FADD _x	914	FSUB _x	1016
FBLD	917	FTST	1022
FBSTP	919	FUCOMI _x	1026
FCHS	922	FUCOM _x	1024
FCLEX _x	924	FWAIT	1029
FCMOV _{cc}	926, 1153	FXAM	1031
FCOMI _x	931	FXCH	1033
FCOM _x	928	FXRSTOR	398, 709, 1035
FCOS	934	FXSAVE	400, 711, 1037
FDECSTP	936	FXTRACT	1039
FDIVR _x	941	FYL2X	1041
FDIV _x	938	FYL2XP1	1043
FEMMS	702	G	
		general-purpose registers	28

H

HLT 1053

I

IDIV 128

IGN xxv

immediate operands 22, 1164

IMUL 131

IN 134

INC 17, 135, 1199

index field 1161

indirect xxv

instructions

128-bit media 291, 1207

3DNow!™ 681, 1205

64-bit media 681, 1207

byte order 1

effects on rFLAGS 1247

formats 1

general-purpose 57, 1207

invalid in 64-bit mode 1195

invalid in long mode 1195

MMX™ 681, 1205

opcodes 19, 1123

origins 1203

reassigned in 64-bit mode 1195

SSE 291, 682, 1206

SSE-2 291, 1206

subsets 25, 1203

system 1047, 1207

x87 909, 1205, 1207

INSx 137

INT 140

INT 3 1054

interrupt vectors 39

INTO 147

INVD 1060

INVLPG 1061, 1141

J

Jcc 15, 150, 1137

JCXZ 15, 154

JECXZ 15

JMP 15, 155

JRCXZ 15

L

LAHF 161

LAR 1068

LDMXCSR 402

LDS 162

LEA 164

LEAVE 166

legacy mode xxvi

legacy x86 xxvi

LES 167

LFENCE 169

LFS 170

LGDT 1071

LGS 172

LIDT 1073

LLDT 1075

LMSW 1077

LOCK prefix 10

LODSx 174

long mode xxvi

LOOPcc 15

LOOPx 176

LSB xxvi

lsb xxvi

LSL 178

LSS 181

LTR 1078

M

mask xxvi

MASKMOVDQU 404

MASKMOVQ 713

MAXPD 406

MAXPS 408

MAXSD 411

MAXSS 414

MBZ xxvii

MFENCE 183

MINPD 417

MINPS 419

MINSD 422

MINSS 425

mod field 1138, 1156

modes 1201

16-bit xxii

32-bit xxii

64-bit xxiii, 1201

compatibility xxiii, 1201

legacy xxvi

long xxvi, 1201

protected xxviii

real xxviii

virtual-8086 xxix

ModRM byte 19, 20, 24, 1138, 1144, 1153

moffset xxvii

MOV 15, 184

MOVAPD 428

MOVAPS 430

MOVD 188, 432, 715

MOVDQ2Q.....	435, 718	group 16	1141
MOVDQA.....	437	group 1a	1139
MOVDQU.....	439	group 2	1139
MOVHLPS	441	group 3	1139
MOVHPD	443	group 4	1139
MOVHPS	445	group 5	1139
MOVLHPS	447	group 6	1139
MOVLPD	449	group 7	1140
MOVLPS	451	group 8	1140
MOVMSKPD.....	191, 453	group 9	1140
MOVMSKPS	193, 455	group P	1141
MOVNTDQ	457	groups	1138
MOVNTL	195	ModRM byte	1138
MOVNTPD.....	459	one-byte.....	1126
MOVNTPS	461	two-byte.....	1128
MOVNTQ	720	x87	1144
MOVQ	463, 722	operands	
MOVQ2DQ.....	465, 724	encodings	1153
MOVSD	467	immediate	22, 1164
MOVSS	469	size	5, 1163, 1164, 1197
MOVSX	199	OR.....	212
MOV _{Sx}	196	ORPD.....	487
MOV _{SXD}	201	ORPS	489
MOVUPD	471	OUT	215
MOVUPS	473	OUT _{Sx}	216
MOVZX	203	overflow.....	xxvii
MSB	xxvii	P	
msb	xxvii	packed	xxvii
MSR	xxxix	PACKSSDW.....	491, 726
MUL	205	PACKSSWB	493, 728
MULPD	475	PACKUSWB	495, 730
MULPS	478	PADDB	497, 732
MULSD	481	PADDD	499, 734
MULSS.....	484	PADDQ	501, 736
N		PADDSB	503, 738
NEG.....	207	PADDSW	505, 740
NOP.....	209, 1199	PADDUSB	507, 742
NOT	210	PADDUSW	509, 744
notation	41, 1123	PADDW	511, 746
O		PAND	513, 748
octword	xxvii	PANDN	515, 750
offset	xxvii, 22	PAVGB	517, 752
opcodes	19	PAVGUSB.....	754
3DNow!™	1141	PAVGW	519, 756
group 1	1139	PCMPEQB.....	521, 758
group 10	1140	PCMPEQD	523, 760
group 11	1140	PCMPEQW.....	525, 762
group 12	1140	PCMPGTB.....	527, 764
group 13	1140	PCMPGTD.....	529, 766
group 14	1140	PCMPGTW.....	531, 768
group 15	1140	PC-relative addressing.....	23

1257

eAX-eSP	xxx	SHUFPS	636
eFLAGS.....	xxx	SIB byte.....	19, 21, 24, 1159
eIP	xxx	SQRTPD	639
encodings.....	17	SQRTPS.....	641
general-purpose	28	SQRTSD	644
MMX	34	SQRTSS.....	646
r8-r15.....	xxxi	SSE	xxix
rAX-rSP.....	xxxi	SSE-2	xxix
rFLAGS.....	xxxii, 1137, 1153, 1247	STC	272
rIP.....	xxxii	STD	273
segment.....	30	sticky bits.....	xxix
system	37	STMXCSR.....	648
x87	36	STOSx.....	274
XMM	31	SUB.....	276
relative.....	xxviii	SUBPD.....	650
REPx prefixes	10	SUBPS	653
RET	15, 239	SUBSD.....	656
REX prefixes.....	14, 24, 1153	SUBSS	659
REX.B bit	16, 45, 1156, 1160	SWAPGS.....	1141
REX.R bit	16, 1155	syntax	41
REX.W bit	16	SYSCALL	1102
REX.X bit	16	SYSENTER	1107
rFLAGS conditions codes.....	1137, 1153	SYSEXIT	1109
rFLAGS register.....	xxxii, 1247	SYSRET	1111
rIP register	xxxii	system data structures.....	38
RIP-relative addressing.....	xxviii, 23	T	
ROL	243	TEST	279
ROR.....	245	TSS.....	xxix
rotate count.....	1164	U	
RSM	1087	UCOMISD	662
RSQRTPS	630	UCOMISS.....	665
RSQRTSS.....	632	underflow.....	xxix
S		UNPCKHPD.....	668
SAHF	247	UNPCKHPS	670
SAL	248	UNPCKLPD	672
SAR	250	UNPCKLPS.....	674
SBB.....	253	V	
scale field	1161	vector.....	xxix
SCASx	256	virtual-8086 mode	xxix
segment prefixes.....	9, 1200	W	
segment registers.....	30	WAIT.....	1045
set.....	xxviii	X	
SETcc	259, 1137	XADD	281
SFENCE.....	261, 1141	XCHG	283
SGDT.....	1088	XLATx	285
shift count	1164	XOR.....	287
SHL	262	XORPD.....	676
SHLD	264	XORPS	678
SHR	267		
SHRD	269		
SHUFPD	634		

Z	
zero-extension	1164

